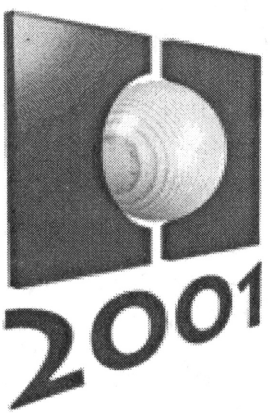
13. mezinárodní matematická olympiáda v informatice

In: Leo Boček (editor); Karel Horák (editor); Tomáš Pitner (editor); Jaromír Šimša (editor); Jaroslav Švrček (editor); Pavel Töpfer (editor): 50. ročník matematické olympiády na středních školách. Zpráva o řešení úloh ze soutěže konané ve školním roce 2000/2001. 42. mezinárodní matematická olympiáda. 13. mezinárodní olympiáda v informatice. (Czech). Praha: Jednota českých matematiků a fyziků, 2001. pp. 165–179.

Persistent URL: http://dml.cz/dmlcz/405036

13. mezinárodní olympiáda v informatice

V pořadí třináctý ročník mezinárodní olympiády v informatice IOI 2001 (International Olympiad in Informatics) se konal ve dnech 14.–21. 7. 2001 ve městě Tampere ve Finsku. Soutěž IOI je pořádána pod záštitou UNESCO a je nejmladší mezinárodní předmětovou olympiádou středoškoláků. Její ohlas ve světě však v posledních letech stále narůstá a každoročně se zvyšuje počet zúčastněných zemí. Letos stoupl jejich počet na 74 a očekává se, že v příštím roce jich bude již kolem osmdesáti.



Reprezentační družstvo České Republiky pro IOI 2001 bylo vybráno na základě výsledků celostátního kola jubilejního 50. ročníku Matematické olympiády — kategorie P. Družstvo mělo následující složení: *Pavel Čížek* (student gymnázia v Kralupech nad Vltavou), *Roman Krejčík* (absolvent gymnázia Ch. Dopplera v Praze 5), *Marek Sulovský* (absolvent gymnázia na tř. Kpt. Jaroše v Brně) a *Miloslav Trmač* (absolvent Biskupského gymnázia v Brně). Vedením družstva byli pověřeni doc. *Pavel Töpfer* a Mgr. *Martin Mareš*, oba z Matematicko-fyzikální fakulty Univerzity Karlovy v Praze.

Stejně jako v minulých letech byla vlastní soutěž rozdělena do dvou soutěžních dnů, v každém z nich řešili soutěžící tři úlohy. Letošní soutěžní úlohy byly algoritmicky mimořádně obtížné, v soutěži se objevily také některé netradiční formy úloh. K nim patřily interaktivní úlohy, v nichž program nemá k dispozici všechna vstupní data najednou, ale dostává je postupně a musí na ně průběžně reagovat (typickým příkladem je program hrající nějakou hru a reagující na tahy protivníka). Jinou netradiční úlohou byla tzv. úloha s otevřeným vstupem, v níž se netestoval vytvořený program, ale ověřovala se správnost výsledků, kterých program dosáhl pro předem známá vstupní data. Velkou novinkou letošního ročníku IOI byla zásadní modernizace prostředí, v němž studenti při soutěži

pracovali a v němž také byly jejich výsledky vyhodnocovány. Namísto dosud v IOI užívaných DOSovských kompilátorů Turbo Pascal a Borland C/C++ a jejich integrovaného vývojového prostředí byl poprvé při soutěži použit operační systém Linux, překladače Free Pascal a GNU C/C++, vývojové prostředí RHIDE. Alternativně bylo možné pracovat také pod operačním systémem Windows. Svá řešení úloh soutěžící odevzdávali k vyhodnocení prostřednictvím nově vyvinutého webového rozhraní. Vyhodnocení se provádělo plně automaticky pomocí předem připravených sad testovacích dat. Různá kvalita navržených algoritmů byla odlišena na základě dosti přísně nastavených časových limitů.

Mezi 272 účastníků bylo rozděleno 23 zlatých, 45 stříbrných a 68 bronzových medailí. Některou z medailí tak podle regulí IOI získala přibližně polovina soutěžících. Výkony našich studentů v soutěži byly poměrně dobré a jejich výsledky nás řadí kolem 20. místa v celkovém pořadí. Olympiáda IOI je ovšem výhradně soutěží jednotlivců a oficiální výsledky družstev se zde vůbec nevyhlašují (neexistuje ani žádná metodika, jak je určovat — zda podle počtu získaných bodů, medailí nebo třeba pořadí). Nejúspěšnějšími zeměmi v letošní IOI byly Slovensko a Singapur se dvěma zlatými a dvěma stříbrnými medailemi, naši získali jednu stříbrnou a dvě bronzové medaile.

Výsledky našich studentů:

36.	Pavel Čížek	344 bodů	stříbrná medaile
85.	Miloslav Trmač	237 bodů	bronzová medaile
100.	Marek Sulovský	197 bodů	bronzová medaile
150.	Roman Krejčík	127 bodů	

Příští 14. ročník mezinárodní olympiády v informatice IOI 2002 se bude konat ve dnech 18.–25. 8. 2002, jeho hostitelem bude Korea. Soutěž bude probíhat na univerzitě Kyung Hee ve městě Young-in, asi 40 km jižně od hlavního města Soulu. Korejští pořadatelé příštího ročníku olympiády během jednání probíhajících ve Finsku pozvali všechny zúčastněné země k účasti na IOI 2002. Mezinárodní výbor IOI také rozhodl, že další budoucí ročníky IOI budou hostit po řadě USA (2003), Řecko (2004) a Polsko (2005).

1. Mobilní telefony

Síť mobilních telefonů čtvrté generace v oblasti poblíž Tampere pracuje takto: celá oblast je rozdělena na čtverce tvořící matici $S \times S$. Řádky a sloupce matice jsou číslovány od 0 do $S - 1$. V každém čtverci je umístěna jedna základnová stanice. Počet aktivních mobilních telefonů nacházejících se v jednotlivých čtvercích se mění, jak se jednotlivé telefony pohybují mezi čtverci, případně se vypínají a zapínají. Každá základnová stanice čas od času oznamuje hlavní ústředně, jak se počet telefonů v jejím čtverci změnil oproti předchozímu ohlášenému stavu.

Napište program pro ústřednu, který bude tyto zprávy přijímat a na základě těchto informací odpovídat na dotazy, jaký je právě počet aktivních telefonů na zadaném území obdélníkového tvaru.

Vstup a výstup. Program čte vstupní data ve formě celých čísel ze svého standardního vstupu a zapisuje výsledky opět ve formě celých čísel do svého standardního výstupu. Vstup má následující tvar: každý řádek obsahuje jeden příkaz, první číslo na řádku určuje typ příkazu, zbývající čísla jsou jeho parametry.

Příkaz	*Parametry*	*Význam*
0	S	Inicializuje matici tak, že její velikost bude $S \times S$ a bude obsahovat samé nuly. Tento příkaz bude použit právě jednou, a to jako první v celém vstupu.
1	$X\ Y\ A$	Přičte A k počtu aktivních telefonů ve čtverci (X, Y). A může být jak kladné, tak záporné.
2	$L\ B\ R\ T$	Dotaz na celkový počet aktivních telefonů ve všech čtvercích (X, Y) takových, že $L \leqq X \leqq R$, $B \leqq Y \leqq T$.
3		Ukončí program. Tento příkaz bude použit právě jednou, a to jako poslední v celém vstupu.

Parametry příkazů budou vždy korektní, není důvod je jakkoliv kontrolovat. Pokud bude A záporné, můžete předpokládat, že počet aktivních telefonů v daném čtverci nikdy neklesne pod nulu. Čtverce jsou indexovány od nuly, tzn. pro matici velikosti 4×4 bude $0 \leqq X \leqq 3$ a $0 \leqq Y \leqq 3$.

Program nijak neodpovídá na vstupní řádky s příkazy 0, 1, ani 3. Po každém řádku s příkazem 2 program vypíše na výstup jeden řádek obsahující jedno celé číslo, které je odpovědí na daný dotaz.

Návod k programování vstupu a výstupu. Používáte-li ke komunikaci standardní prostředky jazyka, postupujte podle následujících příkladů. V těchto příkladech integerová proměnná `last` obsahuje poslední číslo načtené ze vstupu a integerová proměnná `answer` odpověď vašeho programu.

Programujete-li v C++ a používáte `iostreams`, čtěte vstup a zapisujte výstup následovně:

```
cin>>last;
cout<<answer<<endl<<flush;
```

Programujete-li v C nebo C++ a používáte `scanf` a `printf`, postupujte takto:

```
scanf("%d", &last);
printf("%d\n", answer); fflush(stdout);
```

Programujete-li v Pascalu, použijte:

```
Read(last); ... Readln;
Writeln(answer);
```

Příklad

stdin	stdout	vysvětlení
`0 4`		Vytvoří prázdnou matici velikosti 4×4.
`1 1 2 3`		Zvýšení hodnoty na $(1,2)$ o 3.
`2 0 0 2 2`		Dotaz na celkový počet v obdélníku $0 \leqq X \leqq 2$, $0 \leqq Y \leqq 2$.
	`3`	Odpověď na dotaz.
`1 1 1 2`		Zvýšení hodnoty na $(1,1)$ o 2.
`1 1 2 -1`		Snížení hodnoty na $(1,2)$ o 1.
`2 1 1 2 3`		Dotaz na celkový počet v obdélníku $1 \leqq X \leqq 2$, $1 \leqq Y \leqq 3$.
	`4`	Odpověď na dotaz.
`3`		Ukončení programu.

Omezení

Velikost matice	$S \times S$	$1 \times 1 \leqq S \times S \leqq 1\,024 \times 1\,024$
Všechny hodnoty ve čtvercích po celou dobu výpočtu	V	$0 \leqq V \leqq 2^{15} - 1 = 32\,767$

Změny počtu aktivních telefonů	A	$-2^{15} \leqq A \leqq 2^{15} - 1$
Počet příkazů na vstupu	U	$3 \leqq U \leqq 60\,002$
Maximální počet telefonů v celé matici	M	$M = 2^{30}$

Z 20 vstupů zadaných při testování úlohy jich 16 bude používat matici velikosti nejvýše 512 × 512.

2. Pětadvacetština

Tajné zprávy, které si posílá Děda Mráz se svými pomocníčky, jsou obvykle psány ve zvláštním jazyce, kterému říkají pětadvacetština, možná právě proto, že se zapisuje v 25-znakové abecedě. Tato abeceda je podobná latince s jednou jedinou výjimkou: chybí v ní písmeno `Z`. Abeceda tedy obsahuje 25 písmen latinky od `A` do `Y` ve stejném pořadí. Každé slovo 25-štiny je tvořeno právě 25 různými písmeny. Slovo si můžeme zapsat po řádcích do tabulky 5 × 5 — např. slovo `ADJPTBEKQUCGLRVFINSWHMOXY` se zapíše takto:

```
A D J P T
B E K Q U
C G L R V
F I N S W
H M O X Y
```

Přípustná slova 25-štiny jsou právě ta, u nichž jsou písmena v každém řádku i v každém sloupci tabulky uspořádána vzestupně. Tedy slovo `ADJPTBEKQUCGLRVFINSWHMOXY` je přípustné, zatímco slovo `ADJPTBEGQUCKLRVFINSWHMOXY` (liší se prohozením písmen `G` a `K`) přípustné není (vzestupné pořadí písmen je porušeno ve druhém i třetím sloupci).

Děda Mráz má slovník, který obsahuje právě všechna přípustná slova 25-štiny uspořádaná vzestupně (lexikograficky). Slova ve slovníku jsou očíslována pořadovými čísly od 1. Např. slovo `ABCDEFGHIJKLMNOPQRSTUVWXY` má číslo 1 a slovo `ABCDEFGHIJKLMNOPQRSUTVWXY` (`T` a `U` prohozena) má číslo 2.

Slovník je však veliký, převeliký. Napište proto program, který bude umět nalézt ke slovu jeho pořadové číslo a naopak pro zadané pořadové číslo vypsat, kterému slovu toto číslo náleží. Slov ve slovníku určitě není více než 2^{31}.

Vstup. Vstupní soubor se jmenuje `twofive.in` a je tvořen vždy dvěma řádky. První řádek obsahuje jediný znak — `W` nebo `N`. Pokud je

to `W`, pak druhý řádek obsahuje jedno přípustné slovo 25-štiny (tj. řetězec o 25 znacích); pokud je na prvním řádku `N`, potom druhý řádek obsahuje pořadové číslo nějakého přípustného slova 25-štiny.

Výstup. Výstupní soubor se jmenuje `twofive.out` a je jednořádkový. Pokud druhý řádek vstupního souboru obsahoval slovo 25-štiny, obsahuje výstupní soubor pořadové číslo tohoto slova. Jestliže druhý řádek vstupního souboru obsahoval pořadové číslo, potom je obsahem výstupního souboru odpovídající slovo 25-štiny.

Příklady vstupu a výstupu

```
twofive.in
   W
   ABCDEFGHIJKLMNOPQRSUTVWXY
twofive.out
   2

twofive.in
   N
   2
twofive.out
   ABCDEFGHIJKLMNOPQRSUTVWXY
```

3. Gameska

Byla jednou jedna hra pro dva hráče, která se hrála na kruhové desce se sedmi jamkami rozmístěnými podél jejího okraje. Mimo to, každý z hráčů má svůj bank. Hra začíná náhodným rozmístěním dvaceti kuliček do jamek tak, že v každé jamce budou nejméně 2 a nejvýše 4 kuličky. Oba hráčí se v tazích pravidelně střídají. Hráč na tahu si zvolí nějakou neprázdnou jamku, vezme z ní do ruky všechny kuličky a dokud mu nějaké kuličky v ruce zbývají, obchází jamky jednu po druhé ve směru hodinových ručiček počínaje jamkou bezprostředně následující po té, kterou si vybral. Přitom provádí následující akce:

- Má-li v ruce více než jednu kuličku: Pokud jamka, u které stojí, již obsahuje 5 kuliček, vezme si z ní jednu kuličku do svého banku. V opačném případě jednu kuličku ze své ruky do jamky vloží.
- Má-li v ruce jedinou kuličku: Pokud jamka, u které stojí, obsahuje nejméně 1 a nejvýše 4 kuličky, pak všechny kuličky z jamky i tu, kterou drží v ruce, přesune do svého banku. V opačném případě (jamka je prázdná nebo s pěti kuličkami) vloží kuličku, kterou drží v ruce, do soupeřova banku.

Hra končí, jakmile se všechny jamky vyprázdní. Zvítězí hráč, který má v banku více kuliček.

Pro hráče, který táhne jako první, vždy existuje vítězná strategie. Napište program, který hraje tuto hru jako první hráč a ve hře zvítězí. Protihráč použitý při testování hraje optimálně, tzn. jakmile může, vyhraje.

Vstup a výstup. Program čte vstupní data ze standardního vstupu a zapisuje výsledky do standardního výstupu. Váš program je hráč 1, soupeř je hráč 2. Nejprve musí váš program načíst jeden řádek se 7 celými čísly $p_1, \ldots, p_7$, což jsou počáteční počty kuliček v jednotlivých jamkách. Jamky na desce jsou očíslovány od 1 do 7 ve směru hodinových ručiček.

Hra poté začíná s tímto obsahem jamek a prázdnými banky obou hráčů. Váš program hraje následovně:

- Je-li váš program na tahu, vypíše číslo jamky, kterou si zvolil pro svůj tah.
- Je-li na tahu soupeř, váš program načte číslo jamky, kterou si soupeř zvolil (tj. té, u které soupeř svůj tah začíná odebráním kuliček).

Pomůcka. Máte k dispozici program (pod Linuxem `ioiwari2`, pod Windows `ioiwari2.exe`), který z jednoho počátečního rozložení kuliček do jamek hraje hru optimálně jako hráč 2. Po spuštění vypíše na svůj standardní výstup první řádek očekávaný vaším programem; na tomto řádku je popsáno počáteční rozložení kuliček ve tvaru `4 3 2 4 2 3 2`.

Poté program hru hraje, ze svého vstupu čte tahy prvního hráče a do svého výstupu na ně odpovídá tahy svými, přičemž celý dialog ukládá do souboru `ioiwari.out`. Můžete si například `ioiwari2` spustit v jednom okně a váš program v druhém a tahy mezi okny přenášet ručně.

Návod k programování vstupu a výstupu. Používáte-li ke komunikaci standardní prostředky jazyka, postupujte podle následujících příkladů. V těchto příkladech integerová proměnná `last` obsahuje poslední číslo načtené ze vstupu a integerová proměnná `mymove` obsahuje váš tah.

Programujete-li v C++ a používáte `iostreams`, čtěte vstup a zapisujte výstup následovně:

```
cout<<mymove<<endl<<flush;
cin>>last;
```

Programujete-li v C nebo C++ a používáte `scanf` a `printf`, postupujte takto:

```
printf("%d\n", mymove); fflush(stdout);
```

```
scanf("%d", &last);
```

Programujete-li v Pascalu, použijte:

```
Writeln(mymove);
Readln(last);
```

Příklad. Následující tabulka ukazuje jeden z možných průběhů hry o šesti tazích:

	Obsah jamek a banků po provedení tahu								
	1.	2.	3.	4.	5.	6.	7.	Bank1	Bank2
Začátek hry	4	3	2	4	2	3	2	0	0
Tah 1. hráče: 2	4	0	3	5	0	3	2	3	0
Tah 2. hráče: 3	4	0	0	4	1	4	0	3	4
Tah 1. hráče: 5	4	0	0	4	0	0	0	8	4
Tah 2. hráče: 4	0	0	0	0	1	1	1	8	9
Tah 1. hráče: 5	0	0	0	0	0	0	1	10	9
Tah 2. hráče: 7	0	0	0	0	0	0	0	11	9

Hodnocení. Při testování váš program za každou vyhranou hru získá 4 body, za každou remízu 2 body a za prohrané hry 0 bodů.

4. Dvojšifra

Nový šifrovací algoritmus AES (Advanced Encryption Standard) pracuje se třemi bloky velikosti 128 bitů. Pro daný blok zprávy p (původní text) a blok klíče k spočte šifrovací funkce E tohoto algoritmu blok c (zašifrovaný text):

$$c = E(p, k).$$

Standard AES rovněž definuje dešifrovací funkci D, která je inverzní k E:

$$D\big(E(p,k),k\big) = p, \quad E\big(D(c,k),k\big) = c.$$

Šifru AES je možné zdvojit, čímž vznikne šifra Double AES používající dva nezávislé klíče k_1 a k_2. Double AES zprávu nejprve zašifruje pomocí k_1 a výsledek ještě pomocí k_2:

$$c_2 = E\big(E(p,k_1),k_2\big).$$

Vaším úkolem je zjistit klíče k_1 a k_2 pro některé konkrétní zprávy zašifrované algoritmem Double AES. Vždy dostanete jak původní text p, tak odpovídající zašifrovaný text c_2. Navíc víte, že nenulové bity se v obou

klíčích (k_1 i k_2) mohou vyskytovat pouze v nejvyšších $4 \cdot s$ bitech bloku; ostatní bity obou klíčů jsou nulové. Parametr s je součástí vstupu.

Máte k dispozici knihovnu implementující funkce E a D (šifrovací a dešifrovací algoritmus AES).

Odevzdáváte nalezené klíče, nikoliv programy!

Vstup. Máte připraveny soubory `double1.in` až `double10.in`, každý z nich obsahuje jedno konkrétní zadání pro tuto úlohu. Každý vstupní soubor sestává ze tří řádků. Na prvním z nich je celé číslo s, na druhém blok p původního textu a na třetím blok c_2 získaný z p zašifrováním algoritmem Double AES. Oba bloky jsou zapsány jako znakové řetězce, každý z nich je tvořen 32 hexadecimálními ciframi (‚0' až ‚9', ‚A' až ‚F'). Všechna zadání jsou řešitelná.

Součástí knihovny jsou rovněž funkce pro převod bloků do jejich hexadecimálního zápisu a zpět.

Výstup. Vaším úkolem je odevzdat (submitovat) 10 výstupních souborů odpovídajících jednotlivým vstupním souborům. Každý z těchto souborů bude obsahovat tři řádky: Na prvním řádku bude text

`#FILE double` i

kde i značí číslo příslušného vstupního souboru. Na řádku druhém bude klíč k_1 a na třetím klíč k_2 takové, že $c_2 = E(E(p, k_1), k_2)$. Oba klíče budou opět zapsány jako řetězce 32 hexadecimálních číslic. Pokud má úloha více řešení, uveďte libovolné jedno z nich.

Příklad. V tomto příkladu použijeme vstupní soubor číslo 0.

`double0.in`

```
1
00112233445566778899AABBCCDDEEFF
6323B4A5BC16C479ED6D94F5B58FF0C2
```

Možný výstupní soubor:

```
#FILE double 0
A0000000000000000000000000000000
70000000000000000000000000000000
```

Knihovna. K dispozici je knihovna implementující algoritmy AES jak pro FreePascal (Linux: `aeslibp.p`, `aeslibp.ppu`, `aeslibp.o`; Windows: `aeslibp.p`, `aeslibp.ppw`, `aeslibp.ow`), tak pro `GNU C/C++` (Linux i Windows: `aeslibc.h`, `aeslibc.o`) a příklady použití těchto knihoven

(`aestoolp.pas` resp. `aestoolc.c`). Interface obou knihoven je popsán v následujících řádcích.

```
type
   HexStr = String [ 32 ]; {only '0'..'9', 'A'..'F' }
   Block = array [ 0..15 ] of Byte; {128 bits }

procedure HexStrToBlock ( const hs: HexStr; var b: Block );
procedure BlockToHexStr ( const b: Block; var hs: HexStr );
procedure Encrypt ( const p, k: Block; var c: Block );
   {c = E(p,k) }
procedure Decrypt ( const c, k: Block; var p: Block );
   {p = D(c,k) }
```

```
typedef char HexStr[33]; /* '0'..'9', 'A'..'F', '0'-terminated */
typedef unsigned char Block[16]; /* 128 bits */

void hexstr2block ( const HexStr hs, /* out-param */ Block b );
void block2hexstr ( const Block b, /* out-param */ HexStr hs );
void encrypt ( const Block p, const Block k, /* out-param */ Block c );
   /* c = E(p,k) */
void decrypt ( const Block c, const Block k, /* out-param */ Block p );
   /* p = D(c,k) */
```

Omezení. Počet s použitých hexadecimálních číslic klíčů je vždy alespoň 1 a nejvýše 5.

Nápověda. Existuje řešení, které libovolné zadání s tímto omezením na velikost s vyřeší do 10 sekund.

5. Skladiště

Jistá nejmenovaná finská technologická společnost má veliké obdélníkové skladiště, jehož jedinými dvěma zaměstnanci jsou skladník a ředitel skladiště. Stranám skladiště budeme říkat (po směru hodinových ručiček) levá, horní, pravá a dolní. Celá plocha skladiště je rozdělena na stejně velké čtverce, které jsou uspořádány do řad a sloupců. Řady jsou číslovány shora dolů přirozenými čísly 1, 2, ... ; sloupce zleva doprava opět 1, 2, ...

Ve skladišti jsou umístěny kontejnery označené jednoznačnými identifikačními čísly. Každý kontejner zaujímá plochu jednoho čtverce. Skladiště je tak velké, že celkový počet všech kontejnerů je menší než počet řad i než počet sloupců. Ze skladiště se žádné kontejnery neodvážejí, pouze čas od času přibyde nový.

Skladník umisťuje nové kontejnery následovně, aby si usnadnil jejich pozdější vyhledávání podle identifikačních čísel: Předpokládejme, že nově umisťovaný kontejner má identifikační číslo k. Skladník prochází první řadu zleva a hledá první kontejner s identifikačním číslem větším než k.

Pokud tam žádný takový kontejner není, umístí nový kontejner na konec této řady (hned za dosud poslední kontejner). Pokud takový kontejner l najde, pak na jeho místo uloží kontejner k a kontejner l vloží do následující řady stejným způsobem. Takto postupuje tak dlouho, až uloží nějaký kontejner za poslední kontejner v některé řadě nebo na první místo v dosud prázdné řadě.

Kdyby do prázdného skladiště přivezli postupně kontejnery 3, 4, 9, 2, 5, 1, jejich rozmístění ve skladišti by bylo následující:

```
1 4 5
2 9
3
```

Jednoho dne přišel ředitel skladiště za svým jediným podřízeným. Zde je část jejich rozhovoru:

Ředitel: Přivezli kontejner 5 dříve než kontejner 4?
Skladník: Ne, pane řediteli, to se nemohlo stát!
Ředitel: Ó, vy dokážete podle rozmístění kontejnerů ve skladišti určit, v jakém pořadí je přivezli?
Skladník: No, ... částečně. Tak například kontejnery, které tu máme teď, mohli přivézt v pořadí 3, 2, 1, 4, 9, 5 nebo také 3, 2, 1, 9, 4, 5 nebo čtrnácti dalšími způsoby.
(Ředitel, poněkud zmaten, opouští scénu.)

Vaším úkolem je napsat program, který na základě rozmístění kontejnerů ve skladišti určí všechna možná pořadí, v jakých mohly být do skladiště přidávány.

Vstup. Vstupní soubor se jmenuje `depot.in`. První řádek obsahuje jediné celé číslo R: číslo poslední řady skladiště, ve které je uložen alespoň jeden kontejner. Následujících R řádků souboru popisuje, které kontejnery jsou uloženy v prvních R řadách skladiště. Každý z těchto řádků začíná celým číslem M (počet kontejnerů v příslušné řadě), za ním následuje M dalších celých čísel, což jsou identifikační čísla jednotlivých kontejnerů postupně zleva doprava. Všechna identifikační čísla kontejnerů jsou z rozmezí od 1 do 50 včetně. Celkem není ve skladišti více než 13 kontejnerů.

Výstup. Výstupní soubor se jmenuje `depot.out` a každý jeho řádek popisuje jedno z možných pořadí přivážení kontejnerů do skladiště. Každé pořadí je popsáno N-ticí identifikačních čísel, kde N je celkový počet

kontejnerů přivezených do skladiště. Každé z možných pořadí musí být uvedeno právě jednou.

Příklad 1

```
depot.in
3
3 1 4 5
2 2 9
1 3
depot.out
3 2 1 4 9 5
3 2 1 9 4 5
3 4 2 1 9 5
3 2 4 1 9 5
3 2 9 1 4 5
3 9 2 1 4 5
3 4 2 9 1 5
3 4 9 2 1 5
3 2 4 9 1 5
3 2 9 4 1 5
3 9 2 4 1 5
3 4 2 9 5 1
3 4 9 2 5 1
3 2 4 9 5 1
3 2 9 4 5 1
3 9 2 4 5 1
```

Příklad 2

```
depot.in
2
2 1 2
1 3
depot.out
3 1 2
1 3 2
```

Hodnocení. Pokud výstupní soubor obsahuje nějaké chybné pořadí nebo je prázdný, za tento test nedostanete žádný bod. V opačném případě je počet bodů za test určen takto: Jestliže výstupní soubor obsahuje všechna možná pořadí a každé z nich právě jednou, získáte 4 body; pokud obsahuje alespoň polovinu z možných pořadí a každé právě jednou, získáte 2 body; jestliže obsahuje méně než polovinu možných pořadí nebo se nějaké pořadí opakuje, obdržíte 1 bod.

6. Hra

Hru hrají dva hráči na hracím plánu tvořeném soustavou N políček (očíslovaných od 1 do N), mezi nimiž vedou šipky. Každé políčko patří jednomu z hráčů, kterému budeme říkat vlastník tohoto políčka. Navíc je každému políčku přiřazeno jedno kladné celé číslo (hodnota políčka),

přičemž hodnoty všech políček jsou navzájem různé. Políčko číslo 1 je určeno jako startovní.

Hra se hraje takto: Na začátku mají oba hráči skóre 0 a na startovním políčku leží kamínek. Hráči kamínek v průběhu hry přemisťují mezi políčky; to políčko, na kterém kamínek právě leží, budeme značit c. Každý tah provádí vlastník políčka c. Tah sestává z následujících operací:

- Pokud je hodnota políčka c větší než současné skóre vlastníka políčka c, stává se hodnota políčka c novým skóre vlastníka. Pokud není větší, žádné skóre se nemění.
- Poté vlastník políčka c vybere jednu z šipek vycházejících z tohoto políčka a kamínek přesune na políčko, kam tato šipka vede. (Jestliže cílové políčko tahu patří témuž hráči, bude příští tah provádět opět on.)

Jakmile se kamínek vrátí na startovní políčko, hra končí a vítězí ten hráč, jehož skóre je vyšší.

O hracím plánu je známo, že:

- Z každého políčka vede vždy alespoň jedna šipka.
- Každé políčko p je dosažitelné ze startovního políčka, tzn. existuje posloupnost na sebe navazujících šipek, která vede ze startovního políčka do p.
- Po konečném počtu tahů se při jakékoliv hře dostane kamínek opět na startovní políčko (tzn. hra vždy skončí).

Napište program, který bude hru hrát a pokaždé vyhraje. Soupeř použitý při vyhodnocování hraje optimálně, tzn. když může, vyhraje.

Vstup a výstup. Program čte vstupní data ze standardního vstupu a zapisuje výsledky do standardního výstupu. Váš program je hráč 1, soupeř je hráč 2. Nejprve musí váš program načíst popis hracího plánu. První řádek tohoto popisu obsahuje jediné celé číslo: počet políček N ($1 \leqq N \leqq 1000$). Každý z následujících N řádků obsahuje N celých čísel, j-té číslo na i-tém z těchto řádků je 1, jestliže z políčka i vede šipka na políčko j, v opačném případě je zde uvedena 0. Další řádek vstupu obsahuje N celých čísel — určení vlastníka jednotlivých políček (každé z nich je 1 nebo 2). Následuje opět řádek s N celými čísly — hodnoty jednotlivých políček, všechny v rozsahu 1 až N a všechny navzájem různé.

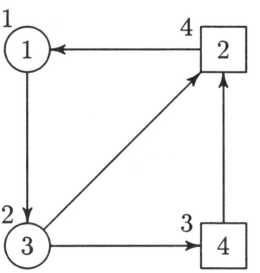
Váš program po načtení popisu hracího plánu hraje následovně:

- Je-li váš program na tahu, vypíše číslo políčka, na které se jeho tahem přesune kamínek.

▷ Je-li na tahu soupeř, váš program načte číslo políčka, na které se kamínek přesunul po soupeřově tahu.

Při testování budou voleny vždy takové hrací plány, na nichž může hráč 1 vyhrát (ať už hru začíná či nikoliv).

Příklad. Hrací plán je vyobrazen na obr. 35. Políčka označená kroužkem patří hráči 1, políčka označená čtverečkem hráči 2. Hodnoty políček jsou zapsány v kroužcích a čtverečcích, čísla políček jsou uvedena u jejich vnějšího okraje.



Obr. 35

Hra může probíhat například takto:

stdin	stdout	vysvětlení
4		*N*
0 1 0 0		Šipky z políčka 1
0 0 1 1		Šipky z políčka 2
0 0 0 1		Šipky z políčka 3
1 0 0 0		Šipky z políčka 4
1 1 2 2		Vlastníci políček
1 3 4 2		Hodnoty políček
	2	Tah hráče 1
	4	Tah hráče 1
1		Tah hráče 2 na startovní políčko — konec hry.

Po skončení této hry má hráč 1 skóre 3 a hráč 2 skóre 2. Hráč 1 tedy zvítězil.

Návod k programování vstupu a výstupu. Používáte-li ke komunikaci standardní prostředky jazyka, postupujte podle následujících příkladů. V těchto příkladech integerová proměnná `target` obsahuje číslo políčka s kamínkem.

Programujete-li v C++ a používáte `iostreams`, čtěte vstup a zapisujte výstup následovně:

```
cin>>target;
cout<<target<<endl<<flush;
```

Programujete-li v C nebo C++ a používáte `scanf` a `printf`, postupujte takto:

```
scanf("%d", &target);
printf("%d\n", target); fflush(stdout);
```

Programujete-li v Pascalu, použijte:

```
Readln(target);
Writeln(target);
```

Pomůcka. Máte k dispozici program (pod Linuxem `score2`, pod Windows `score2.exe`), který si ze souboru `score.in` přečte popis hracího plánu (ve stejném formátu jako má mít vstup vašeho programu) a poté hraje hru jako hráč 2, přičemž jeho výstup odpovídá přesně vstupu vašeho programu a naopak. Své tahy tento program volí náhodně.

Hodnocení. Pro každou z testovaných her získáte plný počet bodů, pokud váš program vyhraje, jinak žádné body. Při vyhodnocování každé hry váš program nejprve hraje s testovacím programem, časový limit je přitom o 1 sekundu vyšší, než je uvedeno v zadání. Průběh hry je zaznamenán. Poté je váš program spuštěn znovu, vstup dostane přesměrovaný ze souboru a je změřena skutečná doba jeho běhu (časový limit podle zadání), přičemž výstup programu musí přesně odpovídat výstupu při prvním testování.