Úloha č. 1
Konfigurační řetězec

Rozmysli, popiš a naprogramuj! 10 b

Tato úloha se skládá ze dvou částí. Tvým úkolem je napsat program a zároveň zdůvodnit proč funguje.

„Vysvětlí mi konečně někdo, o co tady jde?!“

Gandalf se na Froda s pochopením otočil a začal: „Kdysi dávno se rektorát ČVUT rozhodl, že by měl vytvořit informační systém. Doba to byla těžká. Standardů bylo pomálu, architektury procesorů se měnily jak dneska JavaScriptové frameworky a většinu kódu psali fyzici. Bylo vybráno několik místností, ve kterých měla být zřízena potřebná infrastruktura – začalo se jim říkat uzly – a první inženýři začali dávat dohromady první kusy Systému nevědouce, co se z něj stane.

První software pro evidenci studentů byl napsaný v assembleru. Poté se k němu přidal systém na zkoušky v Algolu a systém na zápočty v Cobolu. Pro interakci mezi systémy někdo zbastlil pár řádků v Lispu a už to tak zůstalo. Pak ale bylo potřeba rozšířit systém pro evidenci studentů. Jelikož do assembleru nikdo nechtěl sahat, rozhodli se využít, co funguje, a dodatečné informace neukládat na magnetické pásce, ale skladovat je pomocí trochu Pascalovského kódu na disketě. Spojení těchto dat zařizoval drobný kód v ML. Docenti z FELu ale nadávali, že je to pomalé, tak si vytvořili alternativní agregátor ve Fortranu. A tak si každý napsal svůj skript na to, co zrovna potřeboval, nejlépe u toho použil výsledky cizího skriptu, a vesele ho přidal do systému.

Systém rostl a uzelní inženýři měli více a více práce. Z uzlů je pomalu už nikdo skoro neviděl vycházet. Jenom někdo ze sekretariátu občas donesl řízek s chlebem (ekvivalent kebabu z minulého tisíciletí) k ventilační šachtě a zaklepal. A lidé si začali všímat, že přinesené dary často způsobily, že jejich problém byl do pár hodin vyřešen, zatímco maily kolegů, co řízky nenosili, zůstávaly nezodpovězené celé týdny. Po nějaké době tak oběti na oltář u ventilační šachty nosili téměř všichni. Požadavky ale byly natolik repetetivní, že uzelní inženýři napsali skripty, které požadavky řešily automaticky za ně – včetně konzumace obětí. Jakým způsobem to Systém dělal, nikdo neví.

Po letech ale přišla rekonstrukce a uzly bylo potřeba přesunout. Uzloví inženýři ale protestovali. Zamkli uzly, všechny maily z rektorátu přesměrovali do spamu a na maily ze stavební firmy nechali odpovídat Systém samotný. Jednoho dne ale přijeli dělníci a rozbourali zdi uzlu. Uvnitř ale našli jen prázdnou místnost, ze které byla cítit pryskyřice. Uzel ale fungoval dál. A když byla rekonstrukce hotová, uzelní inženýři byli vidět, jak vchází a vychází z chodby, kde kdysi bývaly dveře do uzlu a kde je dnes jenom učebna. Občas je někdo několik dnů neviděl a pak najenou vyšli z úplně jiné budovy.

Jak uzelní inženýři se Systémem splývali víc a víc, rozhodli se proto rovnou využít Systém k přesouvání mezi uzly. Napsali pár skriptů a začali se mezi uzly pohybovat, jako kdyby byli jenom packety v síti. Toto přesouvání funguje i dnes, jenom postupem času nabobtnalo a navázet spojení mezi uzly je obtížnější. Aragorn nad tím strávil věčnost. Zbytek ti vysvětlí on.“

Zadání

Na uzlu, se kterým se snažíme navázat spojení, běží n skriptů. K navázání spojení je potřeba odeslat uzlu konfigurační řetězec bitů délky m. Tento řetězec poté dostanou všechny skripty na vstupu. Jenomže jsme si všimli, že u většiny řetězců některé skripty spadnou. Vypozorovali jsme, že pokud

změníme i-tý bit řetězce, jistá množina skriptů se začne chovat opačně – pokud padala, tak bude fungovat, a pokud fungovala, tak bude padat. Důležité je, že tato množina zůstane pro daný bit řetězce pořád stejná. Na začátku všechny skripty padají a výchozí konfigurační řetězec sestává ze samých nul. Vaším úkolem je tedy nastavit konfigurační řetězec tak, aby žádný skript nepadal. Také musíte zjistit, kolik různých takových konfiguračních řetězců existuje.

Vstup

Na prvním řádku vstupu se nachází čísla m, n ($m, n \geq 1$) reprezentující délku konfiguračního řetězce a počet běžících skriptů.

Dále následuje m řádků, kde na každém se nachází nejprve číslo k_i určující pro i-tý bit počet skriptů, který ovlivňuje. Dále na tomto řádku následuje k_i čísel, které určují, které skripty jsou tímto bitem ovlivňovány.

Výstup

Na výstupu napište nejprve číslo x reprezentující počet možných konfiguračních řetězců. Pokud je $x > 0$, vypište na další řádek nějaké řešení v podobě konfiguračního řetězce (tedy m bitů).

Ukázka

Vstup

```
2 2
1 1
1 2
```

Výstup

```
1
11
```

Máme konfigurační řetězec délky $m = 2$ a $n = 2$ skriptů. První bit ovlivňuje první skript a druhý bit ovlivňuje druhý skript. Pro konfigurační řetězec jsou čtyři možnosti: `00`, `01`, `10`, `11`. Pro první řetězec budou oba skripty padat, pro druhý a třetí zůstane jeden nefunkční a pro čtvrtý řetězec oba skripty fungují. Je tedy jediné řešení `11`.

Vstup

```
2 3
2 1 3
2 2 3
```

Výstup

```
0
```

Zde máme $n = 3$ skripty a délka konfiguračního řetězce je $m = 2$. Zde je ale vidět, že žádná z možností `00`, `01`, `10`, `11` nezabere jako konfigurační řetězec. Pro řetězec `00` budou všechny tři skripty padat. Pro řetězec `01` bude padat první skript, pro řetězec `10` bude padat druhý skript a pro řetězec `11` bude padat třetí skript.

Vstup

```
3 2
1 1
2 1 2
2 1 2
```

Výstup

```
2
010
```

Zde máme $n = 2$ skripty a délku konfiguračního řetězce $m = 3$. Řešení jsou celkem 2: `010` a `001`.

Bodové podmínky

- Za vyřešení problému jsou 2 **body**.
- Za vyřešení problému polynomiálním algoritmem (v m a n) bez počítání počtu možných řešení je 7 **bodů** (tedy stačí umět říct zda řešení existuje či nikoliv a nějaké vypsat, pokud existuje).
- Za plné vyřešení problému polynomiálním algoritmem je 10 **bodů**.