PRASK
II. ročník, 2015/16
Katedra základov a vyučovania informatiky FMFI UK,
Mlynská Dolina, 842 48 Bratislava

Vzorové riešenia 1. kola letnej časti

vzorák napísal Baklažán
(max. 18 b za riešenie)

1. Perníková chalúpka

Podúlohy a, b, c

Počet možností

Zásadnou otázkou pre prvé tri podúlohy tejto úlohy je "Koľko možností vieme rozlíšiť pomocou X koláčov?". Inými slovami, koľkými rôznymi spôsobmi vie Marienka naukladať koláče na podnos, ak ich má použiť presne X?

Ak má Marienka poslať Jankovi iba jeden koláč, má dve možnosti: buď mu pošle makovník, alebo orechovník. Ak má poslať dva koláče, má dve možnosti, akého druhu bude prvý koláč, a v oboch prípadoch sa ešte môže rozhodnúť, akého druhu bude druhý koláč. Spolu má teda $2 \cdot 2 = 4$ možnosti, ako vybrať koláče: `MM`, `MO`, `OM`, `OO` (kde `M` reprezentuje makovník a `O` orechovník). Ak má Marienka poslať tri koláče, typy prvých dvoch vie určiť štyroma rôznymi spôsobmi (ktoré sú rovnaké, ako keď posielala iba 2 koláče) a pre každý z týchto spôsobov má dve možnosti, čo dať na koniec. Spolu teda má $4 \cdot 2 = 8$ možností (sú to `MMM`, `MMO`, `MOM`, `MOO`, `OMM`, `OMO`, `OOM`, `OOO`). A takto to bude pokračovať ďalej: ak má Marienka poslať 4 koláče, bude mať dvakrát viac možností, než keď mala poslať 3, ak má poslať 5, bude mať dvakrát viac možností než pri štyroch atď.. Všeobecne, pomocou X koláčov vie Marienka rozlíšiť 2^X možností.

To znamená, že pomocou 8 koláčov vie Marienka rozlíšiť $2^8 = 256$ rôznych časov (čo je odpoveď na podúlohu b.). V podúlohe c. potrebujeme rozlíšiť 420 rôznych možností. 8 koláčov nám na to nestačí, 9 koláčov nám dáva $2^9 = 512$ rôznych možností, čo nám už stačí. Na zakódovanie 420 rôznych možností teda potrebujeme minimálne 9 koláčov.

Ako kódovať

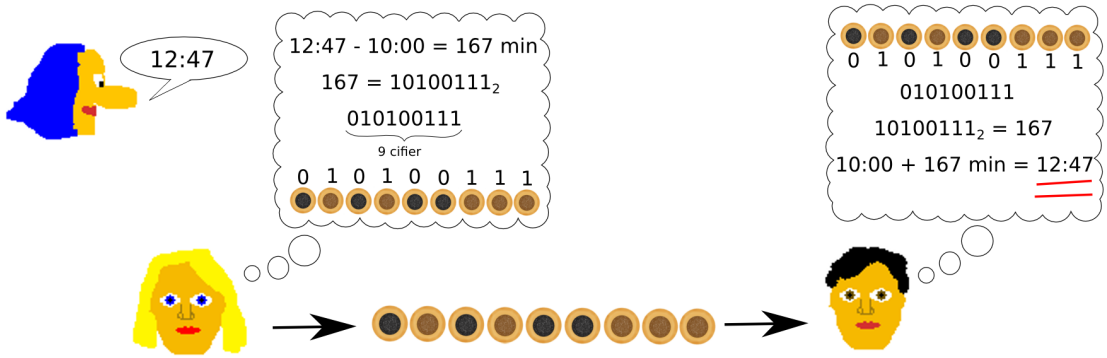
Druhou otázkou je, ako do koláčov zakódovať informáciu o čase. Veľmi jednoduchý prístup, ktorý by bol ale pre Janka a Marienku pomerne pracný, je vypísať si do tabuľky všetky časy, ktoré chceme zakódovať a ku každému z nich priradiť nejakú inú možnosť servírovania koláčov. Tabuľka pre podúlohu a) by teda mohla vyzerať napríklad takto:

čas	koláče	čas	koláče	čas	koláče
11:00	MMMMM	12:20	MOMMM	13:40	OMMMM
11:10	MMMMO	12:30	MOMMO	13:50	OMMMO
11:20	MMMOM	12:40	MOMOM	14:00	OMMOM
11:30	MMMOO	12:50	MOMOO	14:10	OMMOO
11:40	MMOMM	13:00	MOOMM	14:20	OMOMM
11:50	MMOMO	13:10	MOOMO	14:30	OMOMO
12:00	MMOOM	13:20	MOOOM	14:40	OMOOM
12:10	MMOOO	13:30	MOOOO	14:50	OMOOO

Na takejto tabuľke sa Janko s Marienkou dohodnú večer a obaja si ju niekam napíšu (alebo sa ju naučia naspamäť). Keď chce ráno Marienka poslať Jankovi čas Ježibabinho odchodu, nájde v tabuľke riadok s týmto časom a príslušným spôsobom poukladá koláče na podnos. Keď Janko dostane koláče, nájde v tabuľke riadok s takto naukladanými koláčmi a pozrie sa, akému času koláče zodpovedajú.

Tento prístup funguje vždy a je fajn, ak je tabuľka dostatočne malá. Existujú však aj spôsoby, kde si netreba pamätať tabuľku. Jeden z nich je nasledovný:

Všetky časy si očíslujeme zaradom číslami od `0` po `počet časov - 1` (v podúlohe c. by teda číslo i dostal čas i minút po 10:00). Čas číslo i potom Marienka zakóduje tak, že si najprv číslo i napíše v binárnej sústave[1]. Ak malo číslo menej cifier, než je predpísaný počet koláčov, pripíše k číslu zľava toľko núl, aby výsledná postupnosť mala dostatočný počet cifier. Túto postupnosť potom Marienka naukladá na podnos, pričom nulu bude reprezentovať makovníkom a jednotku orechovníkom. Keď Janko dostane koláče, prečíta makovníky ako nuly a orechovníky ako jednotky a takto získané binárne číslo prevedie naspäť do desiatkovej sústavy[2]. Tým získa číslo času, kedy má ujsť.



Všimnite si, že aj tabuľka uvedená ako riešenie podúlohy a. bola v skutočnosti vytvorená týmto spôsobom.

Podúloha d

Ak by Ježibaba s koláčmi nerobila nič, na rozlíšenie 240 možností by sme potrebovali 8 koláčov. Jednoduchý spôsob, ako vyriešiť túto podúlohu pomocou 16 koláčov je poslať Jankovi zakódovaný čas dvakrát. Teda do prvých 8 koláčov Marienka zakóduje požadovaný čas a druhých 8 koláčov bude rovnakých ako prvých 8. Ak Ježibaba niektorý z koláčov vymení za koláč opačného druhu, nebude sa prvá osmica koláčov zhodovať s druhou a podľa toho Janko zistí, že sa niečo stalo. Ak sa prvá a druhá osmica budú zhodovať, Janko bude vedieť, že všetko je v poriadku a z prvej osmice dekóduje čas úteku.

Ako sme spomínali v zadaní, existuje aj riešenie využívajúce iba 9 koláčov. To funguje nasledovne: Marienka do prvých 8 koláčov zakóduje čas a na deviate miesto dá taký koláč, aby bol počet orechovníkov na podnose párny. Ak Ježibaba niektorý koláč vymení za koláč iného druhu, tak buď zmenila makovník za orechovník (čím počet orechovníkov stúpol o 1), alebo orechovník za makovník (čím počet orechovníkov klesol o 1). V každom prípade, po takejto výmene by bol počet orechovníkov nepárny. Keď teda Jankovi prídu koláče, najprv spočíta všetky orechovníky a zistí, či ich je párny počet. Ak áno, tak je všetko v poriadku a z prvých 8 koláčov môže dekódovať čas. Ak nie, potom musela Ježibaba niečo zmeniť.

Podúloha e

Na rozlíšenie 43 200 možností bude Marienka potrebovať aspoň 16 koláčov. Jednoduchým riešením je poslať zakódovaný čas trikrát (použijeme teda $3 \cdot 16 = 48$ koláčov). Ak Ježibaba nejaký koláč zmení, pokazí tým iba jednu z troch kópií Marienkinho kódu, aspoň dve kópie budú stále neporušené. Janko sa teda pozrie na jednotlivé šestnástice koláčov a buď sú všetky tri rovnaké (vtedy z nich jednoducho dekóduje čas), alebo sú dve rovnaké a jedna iná – vtedy vie, že tá jedna iná je zmenená a čas dekóduje zo zvyšných dvoch.

Iné, prefikanejšie riešenie využíva trik, ktorý sme urobili v podúlohe d.: na to, aby sme vedeli zistiť, že Ježibaba niečo zmenila, nám stačí jeden koláč navyše. Do 16 koláčov teda Marienka zakóduje čas a sedemnásty koláč vyberie tak, aby bol počet orechovníkov párny. Následne týchto 17 koláčov ešte raz zopakuje (dokopy teda použije 34 koláčov). Keď Janko dostane koláče, pozrie sa, či sa prvá polovica kolačov zhoduje s druhou. Ak áno, potom vie, že Ježibaba nezmenila druh žiadneho koláča a normálne dekóduje čas. Ak sa sedemnástice nezhodujú, Janko vie povedať, ktorá z nich bola zmenená: je to tá, ktorá obsahuje nepárny počet orechovníkov. Z tej druhej teda môže dekódovať čas.

Podúloha f

To najlepšie nakoniec: riešenie s 25 koláčmi. Ako bolo povedané v zadaní, koláče si tentoraz usporiadame do

[1] http://www.gym-informatika.estranky.cz/clanky/dvojkova-sustava.html
[2] Tento krok môže vynechať, ak je dostatočný macher na to, aby všetko rátal v binánej sústave.

štvorca 5×5. Do 16 koláčov tvoriacich štvorec 4×4 v ľavom hornom rohu nášho štvorca 5×5 Marienka zakóduje čas úteku. V prvých štyroch riadkoch doplní piaty koláč tak, aby v každom z týchto riadkov bol párny počet orechovníkov. Nakoniec doplní koláčmi posledný riadok tak, aby bolo v každom stĺpci párne veľa orechovníkov.

V každom stĺpci aj v každom z prvých štyroch riadkov je teda párny počet orechovníkov. Koľko orechovníkov je v piatom riadku? Keďže v každom stĺpci je párne veľa orechovníkov, aj v celom štvorci bude párne veľa orechovníkov. V prvých štyroch riadkoch je dohromady tiež párny počet orechovníkov (lebo v každom z nich je párny počet), teda aj v piatom riadku ich musí byť párny počet.

Ak Ježibaba vymení nejaký koláč za koláč opačného druhu, v riadku obsahujúcom tento koláč sa počet orechovníkov zmení na nepárny. Rovnako aj v stĺpci obsahujúcom tento koláč. Keď teda Janko dostane koláče, najprv si spočíta, či je v každom riadku párny počet orechovníkov. Ak áno, potom Ježibaba žiaden koláč nevymenila za opačný a Janko môže dekódovať čas. Druhá možnosť je, že v jednom riadku je počet orechovníkov nepárny. Vtedy Janko vie, že Ježibaba zmenila niektorý z koláčov v tomto riadku. Spočíta počty koláčov v jednotlivých stĺpcoch a podľa toho, v ktorom stĺpci je nepárny počet orechovníkov vie, ktorý koláč Ježibaba vymenila. Predstaví si, že by na jeho mieste bol koláč opačného druhu (teda taký koláč, aký tam dala Marienka) a dekóduje čas úteku.

2. Počítače zvnútra

vzorák napísal Žaba
(max. 15 b za riešenie)

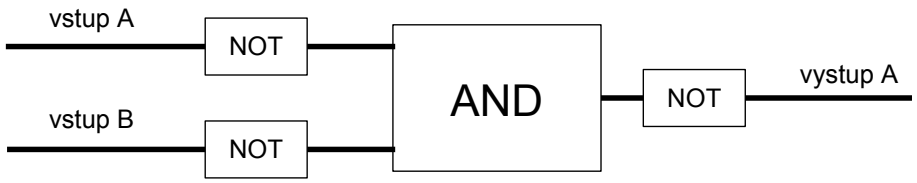
Výraz *znegovať* označuje zmenu hodnoty na opačnú hodnotu, teda použitie jedného hradla NOT. Vo vzorovom riešení budem používať tento výraz, pretože je bežne používaný v matematike.

Podúloha a.

Keď sa snažíme pracovať s logickými hradlami alebo funkciami, častokrát sa nám oplatí napísať si celú tabuľku hodnôt pre hľadané hradlo a hľadať podobnosti s hradlami, ktoré poznáme. Vieme, že hradlo OR má vrátiť nulu iba v tom prípade ak sú obe vstupné hodnoty rovné 0, inak vráti hodnotu 1. To sa trochu podobá na hradlo AND, ktoré vráti 1 iba v jedinom prípade – keď sa obe vstupné hodnoty rovnajú 1.

Ak ale znegujeme výsledok AND, podobnosť s OR je ešte výraznejšia, pretože negovaný AND (tiež NAND) vracia 0 iba v jedinom prípade, presne tak, ako hradlo OR. Akurát tie prípady sú odlišné. OR na to potrebuje dve nuly, ale NAND dve jednotky. Ako teda zaručíme, že ak naše hradlo dostane dve 0, náš znegovaný AND dostane dve 1? Jednoducho obe vstupné hodnoty znegujeme.

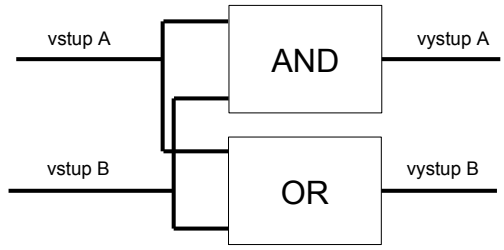
Celé hradlo preto vyzerá nasledovne: znegované vstupné hodnoty vložíme do hradla AND, ktorého výsledok tiež znegujeme. Vyskúšaním všetkých štyroch možností (00, 01, 10 a 11) ľahko overíme, že takéto niečo sa správa presne ako bolo popísané v zadaní.



Podúloha b.

Na vstupe máme dve hodnoty a našou úlohou ich bolo usporiadať – na prvý výstup vrátiť menšiu z nich, na druhý väčšiu. Môžeme teda vyskúšať všetky možnosti ako vyzerá vstup a skúsiť na to napasovať nejaké z hradiel, ktoré poznáme. Oveľa lepšie je však pozrieť sa na to, ako vyzerajú výstupné hodnoty. Menšia z dvoch hodnôt bude takmer vždy hodnota 0. Jediná výnimka je, keď máme na vstupe dve 1. To je ale presne to, čo robí hradlo AND. A väčšia hodnota bude takmer vždy 1, okrem prípadu, keď máme na vstupe dve 0, čo nám vie ošetriť hradlo OR.

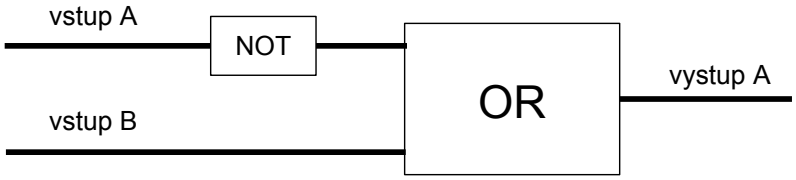
Celé hradlo teda vyzerá nasledovne: obe vstupné hodnoty vložíme do jedného hradla AND, ktoré bude vracať menšiu hodnotu a do jedného hradla OR, ktoré bude vracať väčšiu hodnotu. Uvedomme si, že hradlo AND je v istom zmysle rovnaké ako funkcia minimum a hradlo OR je rovnaké ako maximum.



Podúloha c.

Pri zostrojovaní hradla implikácie použijeme podobný prístup ako v podúlohe a.. Všimnime si, že hradlo IMP vracia 0 iba v jedinom prípade, keď je na prvom vstupe hodnota 1 a na druhom vstupe hodnota 0. To sa podobá na hradlo OR, ktoré vracia 0 tiež iba v jedinom prípade, keď sú obe vstupné hodnoty 0. Preto, ak chceme ako výsledok použiť výsledok hradla OR, musíme upraviť vstupné hodnoty tak, aby sa vstup 10 zmenil na 00, ktorý môžeme posunúť hradlu OR. To znamená, že nám stačí znegovať prvý vstup.

Hradlo teda vyzerá nasledovne: znegovaný prvý vstup a pôvodný druhý vstup vložíme do hradla OR, ktorého výstup je výstupom hradla IMP. Overením štyroch možností ľahko preveríme správnosť takéhoto riešenia.



Podúloha d.

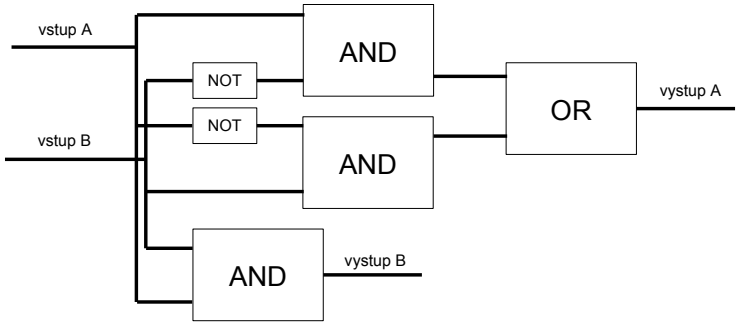
Hradlo XORc je prvé maličké sčitovacie hradlo s dvoma vstupmi a dvoma výstupmi. Na *vystup B* vraciame 1, ak sa prenáša jednotka do vyššieho rádu. Uvedomme si ale, že to nastane iba v prípade, ak sú obe vstupné hodnoty 1. Takže vypočítať druhý vstup vieme pomocou jednoduchého hradla AND.

Komplikovanejšie je to s *vystup A*. Na ten máme vrátiť 1 v prípade, že prvý vstup je 1 **a** druhý vstup vstup je 0 **alebo** je prvý vstup 0 **a** druhý vstup je 1. Logické spojky sme zdôraznili zámerne. Vidíme, že aj v bežnom jazyku ich častokrát používame a keď si problém sformulujeme správnym spôsobom, riešenie sa ukáže samo.

Najskôr musíme vedieť zistiť, či je na prvom vstupe 1 a na druhom vstupe 0, pričom budeme chcieť použiť hradlo AND (a zároveň). To je však problém, s ktorým sme sa stretli už dvakrát, v podúlohách a. a c.. Keďže AND vráti 1 iba ak sú oba jeho vstupy rovné 1, tak na overenie zadanej podmienky vložíme do hradla AND pôvodnú prvú hodnotu a znegovanú druhú hodnotu. To isté, akurát opačne spravíme pri overovaní druhej časti – prvý vstup je 0 a druhý 1. V tomto prípade znegujeme prvý vstup a necháme druhý nezmenený.

Máme teda dve hradlá AND, z ktorých jedno vráti 1, ak je na vstupe jeden z dvoch správnych vstupov. A zistiť, či aspoň jedno z týchto hradiel naozaj vráti 1, vieme pomocou hradla OR (alebo). Tým sme skonštruovali to, čo nám popisovala vyššie uvedená veta.

Celé hradlo dokopy teda vyzerá nasledovne: zoberieme dve hradlá AND. Do prvého z nich vedie prvý vstup a znegovaný druhý vstup. Do druhého vedie znegovaný prvý vstup a pôvodný druhý vstup. Výstupy týchto dvoch hradiel potom vložíme do hradla OR. Jeho výstup bude *vystup A*. Hodnotu *vystup B* zistíme ako AND oboch pôvodných vstupov.

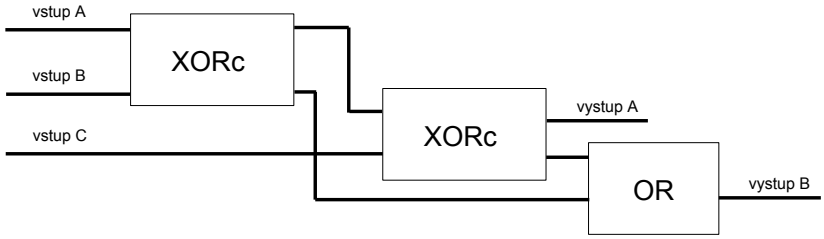


Podúloha e.

Ak sa nám podarilo spraviť dvojvstupový XORc, vytvoriť XOR3c by malo byť pomerne jednoduché. Pomocou XORc sčítame prvé dve vstupné hodnoty. Výsledok tohto sčítania je prvý výstup tohto hradla. K tomuto výstupu teda pričítame tretiu vstupnú hodnotu pomocou druhého hradla XORc. Prvý výstup tohto druhého hradla XORc je naozaj prvý výstup hradla XOR3c. Ostáva nám vyriešiť už len otázku, či nastal prenos do vyššieho rádu.

Prenos však musel nastať pri niektorom z dvoch sčítaní pomocou hradiel XORc. A tie majú špeciálny výstup, na ktorý vrátia 1 ak takáto udalosť nastala. Ak teda aspoň jeden z týchto dvoch výstupov obsahuje hodnotu 1, chceme ju vrátiť aj na výsledný *vystup B*, v čom nám pomôže hradlo OR. Všimnime si, že sa nemôže stať, že obe hradlá XORc vrátia prenosovú hodnotu 1. Súčet troch hodnôt je totiž najviac 3, a pri tom nastáva najviac jeden prenos.

Celé hradlo teda vyzerá: prvé dva vstupy pošleme do hradla XORc. Prvý výstup tohto hradla vložíme spolu s tretím vstupom do druhého hradla XORc. Prvý výstup tohto sčítania vrátime na *vystup A*. Hodnotu *vystup B* získame ako výsledok hradla OR, do ktorého vložíme druhé výstupy oboch hradiel XORc.

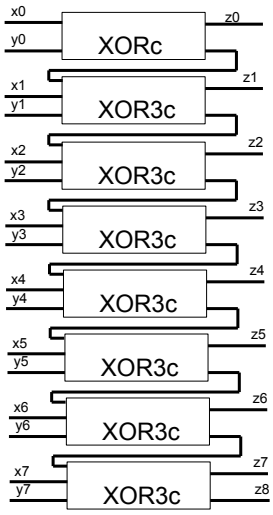


Podúloha f.

Ako iste tušíte, predchádzajúce dve podúlohy pomaly ale iste smerovali k riešeniu poslednej podúlohy – vytvorenie hradla SUM, ktoré dokáže sčítavať binárne čísla. Pozrime sa teda na to, ako sa sčítavajú binárne čísla.

Ako bolo napísané v zadaní, musíme ich sčitovať postupne po cifrách (bitoch), pričom začínať musíme najmenej významnými ciframi. To sú v tomto prípade cifry x_0 a y_0. A sčítať dve hodnoty predsa vieme pomocou hradla XORc. Prvý výstup tohto hradla nám dá súčet príslušných cifier, čo bude hodnota cifry z_0 patriaca výsledku. Druhý výstup udáva, či nastal prenos. Ak nastal prenos, musíme ho totiž zahrnúť do ďalšieho sčitovania. Na zistenie cifry z_1 nám teda nestačí sčítať cifry x_1 a y_1, ale musíme pridať aj prenos z predchádzajúceho sčítania. No a sčítať tri hodnoty vieme pomocou hradla XOR3c.

Celé hradlo SUM teda vyzerá nasledovne: použijeme jedno hradlo XORc a sedem hradiel XOR3c. Každé z týchto hradiel dostane na vstup po jednej cifre z čísel x a y, pričom tieto cifry si musia zodpovedať. Hradlo XORc dostane dvojicu (x_0, y_0), zvyšné dvojice (x_1, y_1), (x_2, y_2) až (x_7, y_7) patria hradlám XOR3c. Tieto hradlá dostanú ako tretí vstup prenosovú hodnotu z predchádzajúceho sčítania, teda (x_1, y_1) sa sčítava s prenosom z (x_0, y_1), dvojica (x_2, y_2) sa sčítava s prenosom z (x_1, y_1) atď.. Prvé výstupy jednotlivých sčítaní sú postupne cifry z_0 až z_7. Cifra z_8 je rovná prenosu posledného sčítania (x_7, y_7).



Funkčné riešenia v Logisime si môžete stiahnuť tu.

3. Písanie si v tajnosti

15 bodov za riešenie

Podúloha a)

V tejto podúlohe ste mali za cieľ vylúštiť text zakódovaný Caesarovou šifrou. Na úspešné vyriešenie existuje viacero možností. Každá z nich ale využíva fakt, že máme len 26 možností pre kľúč. Posunúť abecedu o viac ako 26 písmen totiž nemá zmysel. Posunutím o 27 písmen dostaneme to isté ako posunutím o 1. Nič nám teda nebráni vyskúšať všetky možnosti.

Túto podúlohu zvládneme vyriešiť aj pomocou pera a papiera. Postupne si vyskúšame posunúť správu o niekoľko písmen dozadu, až kým nedostaneme zo správy niečo zmysluplné. Aby sme však nemuseli posúvať celý text, o ktorom aj tak rýchlo zistíme, že nie je správny, stačí nám vybrať si len jedno slovíčko, skúsiť na ňom rôzne hodnoty pre kľúč a potom vybrať tú správnu a pomocou nej rozšifrovať zvyšok správy.

Vyberieme si druhé slovo zo správy, „DZX". Je totiž krátke a bude sa nám na ňom rýchlo zisťovať kľúč.

0	DZX
1	EAY
2	FBZ
3	GCA
4	HDB
5	IEC
6	JFD
7	KGE
8	LHF
9	MIG
10	NJH
11	OKI
12	PLJ
13	QMK
14	RNL
15	SOM

Tu si hneď všimneme slovíčko „SOM". Je to pomerne časté slovenské slovo. Môžme teda skúsiť celú správu posunúť o 15 miest v abecede, čím odhalíme zašifrovaný text.

Pre tých lenivejších, existujú aj menej namáhavé postupy. Napr. napísať si program ktorý vypíše všetkých 26 možností pre originálnu správu, a nájsť si medzi nimi tú, ktorá dáva zmysel. A ešte rýchlejšie riešenie je nájsť si na internete takýto program, napr. http://www.mygeocachingprofile.com/codebreaker.caesarcipher.aspx

Podúloha b)

V tejto podúlohe už máme pre kľúč omnoho viacej možností, a nedokážeme ich vyskúšať všetky. Musíme na to ísť rozumnejšie. Vieme že každé písmeno sa nám zmenilo na nejaké iné. V texte ale máme dostatok informácii na to, aby sme si vedeli tieto pôvodné písmena zistiť. Tu sa nám oplatí využiť `find and replace` funkciu textového editoru, o ktorom bola reč v minulom zašifrovanom texte. Keďže celý zašifrovaný text má iba veľké písmená, ak o niektorom písmene zistíme jeho pôvodný tvar, nahradíme ho v texte malými písmenami. Ak napríklad zistíme, že sa „a" zašifrovalo na „d", nahradíme v texte všetky veľké „D" malým „a". Takto budeme mať prehľad o tom, čo sme už rozšifrovali a čo ešte nie.

Začnime tým úplne najočividnejším, a to sú odkazy. Vidíme text "JYYU://BBB.NWU.WN/", z toho vieme určiť že „J" je „h", „Y" je „t", „U" je „p" a „B" je „w", lebo každý link začína „http://www.". To je dobrý začiatok. Až vymeníme tieto písmena, na prvom riadku správy budeme mať text „DhSR DIDP,". Keďže sa to nachádza na prvom riadku, mohlo by to byť oslovenie. A keďže druhé písmeno oslovenia je „h", môžme si tipnúť, že prvé slovo je „ahoj", a nahradiť príslušné písmenká. Na prvom riadku potom budeme mať „ahoj aIaP". Zo

zadania vieme, že Andrej posiela list Adamovi, takže „aIaP" by mohlo byť v skutočnosti „adam". Na poslednom riadku by mal byť podpis, a je tam „aVdEKj", čo nemôže byť nič iné ako meno odosielateľa „andrej"..

Teraz máme dosť písmen na to, aby sme si zvyšné domysleli z čiastočných slov, ktorých je tam teraz už neúrekom. Celý princíp spočíva v tom, že sa pozeráme na slová v ktorých poznáme väčšinu písmen a vieme odhadnúť, aké sú tie zvyšné. Napr. zo slov ako „poFedat", „pZWmeno", „mZmoMhodom" vieme uhádnuť, že to majú byť slová „povedat", „pismeno" a „mimochodom". Takýmto spôsobom zistíme postupne všetky písmena.

Podúloha c)

Tretia podúloha bola o dosť ťažšia. Na jej riešenie bolo potrebné použiť metódu zvanú **frekvenčná analýza**. Princíp tejto metódy je, že rôzne písmena sa v normálnom texte vyskytujú rozlične veľa krát. Napríklad, asi by sme ťažko našli v našom texte 60 krát písmeno „q", ktoré sa v slovenčine takmer nevyskytuje. Naopak, očakávame, že sa v ňom bude nachádzať naozaj veľa písmen „a" alebo „e".

Toto vieme pri riešení využiť nasledovne. Predstavme si, že sa písmeno „a" zašifrovalo na písmeno „W". To znamená, že v zašifrovanom texte sa nachádza toľko písmen „W", koľko bolo v pôvodnom texte písmen „a". A ak vieme, že „a" je najčastejšie slovenské písmeno a „W" sa v zašifrovanom texte nachádza najväčší počet krát, ľahko usúdime, že „W" je vlastne zašifrované „a".

S dešifrovaním textu môžeme začať tak, že si určíme nejaký typický slovenský[3] text a spočítame, ako často sa v ňom vyskytujú jednotlivé písmená. Pomocou týchto počtov určíme tabuľku frekvencií jednotlivých písmen. Ak ste však vyriešili druhú šifru, získali ste našu frekvenčnú tabuľku (odkaz), ktorá bola uspôsobená na zašifrovaný text. Dala by sa použiť aj vlastná frekvenčná tabuľka, dešifrovanie by však bolo kúsok ťažšie. Následne si zrátame frekvencie v našom zašifrovanom texte. To urobíme napríklad pomocou stránky, ktorú sme vám tiež poskytli v druhej šifre (odkaz).

Môžeme začať s dešifrovaním. Vidíme že písmeno „D" sa nám vyskytuje v zašifrovanom texte najčastejšie, a často sa vyskytuje ako druhé písmeno v dvojpísmenových slovách. Navyše sa často vyskytuje aj osamote. Môžeme teda s vysokou istotou povedať, že „a" sa nám zašifrovalo na „D". Ďalšie písmeno, ktoré má v zašifrovanom texte vysokú frekvenciu je „N". Z tabuľky vidíme, že jeho frekvencia sa blíži frekvencii písmena „e". Môžme si buď tipnúť, že to „e" naozaj je, ale keďže sa blízko vyskytuje aj písmeno „o", môžeme zatiaľ urobiť niečo iné.

Pozrime sa radšej na dvojpísmenové slová, v ktorých na druhom mieste je „a". Všimnime si často sa vyskytujúce dvojice „Ca" a „Ga". Prečo práve tieto? Oboje z nich sa vyskytujú aj samostatne. To znamená, že sa pravdepodobne jedná o písmena „s" a „z", nevieme ale ktoré je ktoré. Toto skombinujeme s tabuľkou frekvencií, a zistíme že „s" má frekvenciu 4.3% zatiaľčo „z" len 2%. V našom texte má „C" frekvenciu 4.8% a „G" len 2.9%. Z toho vyvodíme, že „C" je zašifrované „s" a „G" je zašifrované „z".

Tým sme odhalili ďalšie dvojice, „Xz" a „sX". Mohlo by ísť o slová „az" a „sa", ale „a" už máme dešifrované, teda „X" nemôže byť „a". Jediné písmeno, ktoré tam sedí na obe slová je „u". Písmenami „z" a „s" sme si tiež dokázali hypotézu, že „e" je zašifrované ako „N", pretože sa nám v texte objavilo slovo „zasN", ktoré vylučuje možnosť, že „N" je „o" („zaso" nie je rozumné slovo).

Odhalenie „e" nám môže pomôcť, stačí nájsť dvojpísmenové slovo končiace „e". Nájdeme viacero výskytov „Be". Možnosti sú slová „je", „ze". Keďže ale „z" už máme rozšifrované, „B" musí byť „j".

Môžme si teraz opäť siahnúť do frekvenčnej analýzy, keďže písmeno „T" sa v našom texte vyskytuje s frekvenciou 6.6%, pričom v obyčajnom texte sa „o" vyskytuje s frekvenciou 6.4%. Frekvencia žiadneho zostávajúceho písmena nie je blízko k tomuto číslu, a teda môžme s vysokou istotou povedať, že „T" je „o". Ďalšie písmeno na muške je „E", ktoré je v texte na viacerých miestach samo. Jednopísmenných slov v slovenčine nie je veľa, a už vôbec nie po tom, čo sme odstránili „a", „z", „s", „o". Ostávajú nám len slová „i" a „v". Keby to ale bolo „i", vznikne nám viacero nezmyselných slov ako napríklad „ieAeQ". Takže písmeno „E" bude zašifrované „v".

Ukáže sa nám slovíčko „vojJou", ktoré môže byť jedine slovo „vojnou". Teda „J" je „n". Tiež sa ukazujú napríklad slová „soR" a „RaR" – možné „som" a „mam". Odkryl sa nám kus textu, „na vZmenu na nasom useWu", z ktorého vidíme, že „Z" je „y" a „W" je „k".

Teraz už máme odkrytých dosť slov na to, aby sme rovnakým spôsobom odkryli celý zvyšok textu.

4. Zjedená pizza

vzorák napísal Mário
(max. 15 b za riešenie)

Aby sme vyriešili túto úlohu, stačí nám robiť to, čo robil Adam v rozprávke v zadaní. Začneme od najväčších krabíc – vždy do nich vložíme toľko menších, koľko sa zmestí, pričom zvyšné krabice musia zostať vonku.

Aby sme si v našom riešení udržali poriadok, v jednej premennej `mozem_zabalit` si budeme pamätať, koľko aktuálne spracúvaných krabíc vieme zabaliť do väčších. Môžeme si všimnúť, že aj keď sú v krabici 8×8

[3] Samozrejme, v rôznych jazykoch sú vyskyty písmen rôzne. Napríklad v angličtine sú najčastejšie písmená „e" a „t".

štyri krabice 4×4, stále sa tam zmestí 16 krabíc 2×2. Teda väčšie krabice neovplyvňujú tie menšie, ak sú spoločne v jednej krabici. Na spočítanie, koľko menších krabíc vieme zabaliť, nám preto stačí vynásobiť premennú `mozem_zabalit` štyrmi .

Postupovať budeme od najväčších krabíc po najmenšie a v každom kroku bude našou úlohou "zabaliť krabice s hranou dĺžky 2^i". Zakaždým spočítame, koľko takýchto krabíc musí zostať vonku a koľko miesta bude pre menšie krabice.

Krabíc zabalíme najviac `mozem_zabalit`. Ak sa všetky krabice zmestia do väčších, veľkosť voľného miesta pre ešte menšie krabice bude `4 * mozem_zabalit`. Ak sa niektoré krabice nezmestia dnu, zostávajú vonku a teda ich pripočítame ku `krabice_vonku`. Tiež ale vytvoria nový priestor pre ešte menšie krabice, teda menších krabíc budeme vedieť zabaliť `4 * (mozem_zabalit + ostali_vonku)`.

Krabice jednej veľkosti "zabalíme" v konštantnom čase $O(1)$, teda ak máme k veľkostí krabíc, riešenie bude potrebovať $O(k)$ času. Keďže vstup spracúvame odzadu, musíme ho celý načítať do poľa, teda aj pamäťová zložitosť bude $O(k)$. V tejto úlohe boli ale všetky vstupy s $k = 20$, tak ste mohli tvrdiť, že beh programu nezávisí od veľkosti čísel na vstupe a prehlásiť časovú zložitosť za konštantnú (ale 1. takéto tvrdenie je vždy potrebné vysveliť, 2. časový odhad $O(k)$ je informatívnejší).

Listing programu (C++)

```
#include <iostream>

using namespace std;

int main(){
  int pocty[20];
  for(int i = 0; i < 20; i++)
    cin >> pocty[i];
  int krabice_vonku = 0;
  long long mozem_zabalit = 0;

  for(int i = 19; i >= 0; i--){
    if (pocty[i] > mozem_zabalit){
      krabice_vonku += pocty[i] - mozem_zabalit;
      mozem_zabalit += pocty[i] - mozem_zabalit;
    }
    mozem_zabalit *= 4;
  }
  cout << krabice_vonku << endl;
}
```

Niektorí z vás skúšali aj opačný prístup – vkladať krabice od najmenšej. Často ste si ale nevšimli, že nám vonku ostávajú krabice rôznych veľkostí. Ak totiž príde veľká krabica (napr. taká, do ktorej sa zmestia všetky menšie), počet tých, ktoré zostanú vonku sa nedá vyrátať bez toho, aby sme sa pozreli na každú menšiu velkosť zvlášť.

5. Zázračné karty

vzorák napísal Šandyna
(max. 15 b za riešenie)

Na začiatok je dôležité si všimnúť niekoľko vecí. Prvé pozorovanie je, že keď x kariet presunieme na vrch a potom x kariet presunieme na spodok, budú v rovnakom poradí, ako na začiatku. Podobne môžeme vidieť, že ak presunieme n kariet jedným smerom, budú opäť všetky na pôvodnom mieste.

Predstavme si, že si karty rozložíme do kruhu a medzi vrchnú a spodnú kartu vložíme zarážku tak, aby vrchná karta bola napravo od zarážky. Čo sa stane, ak presunieme vrchnú kartu na spodok balíčka a opäť ich rozložíme do kruhu? Jediné, čo sa zmení je pozícia zarážky, ktorá bude mať naľavo od seba vrchnú kartu, ktorá sa posunula a napravo bude druhá karta zvrchu. To znamená, že zarážka sa nám po kruhu posunula o jedno miesto doprava. A samozrejme, presunutie spodnej karty navrch je len posunutie zarážky o jedno miesto doľava.

Ak teda iba presúvame karty zvrchu naspodok a naopak, stačí si pamätať, kam sme presunuli našu zarážku. A keď máme následne vypísať aktuálny stav balíčka, tak pôjdeme postupne po kruhu kariet, pričom začínať budeme na karte napravo od zarážky (aktuálna vrchná karta) až kým nenarazíme opäť na zarážku.

Čo sa ale stane ak otočíme celý balíček? Vrchná karta bude zrazu naľavo od zarážky, kým spodná bude napravo. Samotné usporiadanie kariet sa teda nezmenilo a dokonca ani zarážka sa neposunula, iba sa zmenila strana, na ktorej je vrchná karta. Je jasné, že ak v takomto stave budeme presúvať vrchnú kartu naspodok, zarážka sa posunie o jedno miesto **doľava**. Takisto výsledok sa bude vypisovať v opačnom smere ako predtým, lebo vrchná karta je na opačnej strane zarážky.

A ako to naprogramujeme? Budeme si pamätať dve premenné. Premenná *reverz* určuje, na ktorej strane od zarážky sa nachádza vrchná karta. Premenná *zarazka* potom hovorí pozíciu zarážky v našom kruhu. My ale nemáme kruh, iba obyčajné pole. To nevadí. Hodnota $zarazka = 0$ bude znamenať, že zarážka sa nachádza medzi prvou a n-tou kartou balíčka. $zarazka = 1$ znamená, že zarážka je medzi prvou a druhou kartou atď. Ak

teda budeme musieť otočiť balíček, jednoducho upravíme hodnotu *reverz* a keď budeme musieť presunúť kartu, tak podľa hodnoty *reverz* zmeníme hodnotu *zarazka* – ak je vrchná karta napravo od zarážky a presúvame túto kartu naspodok, k hodnote *zarazka* pričítame číslo 1, ak je vrchná karta naľavo, tak k hodnote *zarazka* pričítame -1. A naopak pri presúvaní spodnej karty.

Samozrejme, môže sa nám stať, že hodnota *zarazka* bude záporná, alebo privysoká. Vtedy však využijeme naše prvé pozorovanie – ak posunieme n kariet jedným smerom, dostaneme rovnakú situáciu. To znamená, že k hodnote *zarazka* môžeme kedykoľvek pričítač alebo odčítať číslo n bez toho, aby sme zmenili situáciu, ktorú popisujeme. Vďaka tomu bude hodnota *zarazka* vždy v intervale 0 až $n - 1$.

Nakoniec, keď budeme chcieť vypísať výsledok, tak si z hodnôt *reverz* a *zarazka* zistíme, kde sa nachádza vrchná karta a v správnom smere (závislom od *reverz*) vypíšeme postupne všetky čísla. Časová aj pamäťová zložitosť bude lineárna od n, čo zapíšeme ako $O(n)$.

Listing programu (C++)

```cpp
#include<bits/stdc++.h>

using namespace std;

int main() {
  int n, q;
  cin >> n >> q;
  vector<int> karty(n);
  for(int i = 0; i < n; i++) {
    cin >> karty [i];
  }

  char op;
  int reverz = 1, zarazka = 0;
  for(int i = 0; i < q; i++) {
    cin >> op;
    if (op == 'D')
      zarazka -= reverz;
    else if (op == 'H')
      zarazka += reverz;
    else if (op == 'R')
        reverz *= -1;
  }

  if (reverz == -1) {
    reverse(karty.begin(), karty.end());
    zarazka *= -1;
  }

  zarazka = ((zarazka % n) + n) % n;

  for (int i = zarazka; i < n; i++)
    printf("%d%c", karty[i], (zarazka==0 && i==n-1) ? '\n' : ' ');

  for (int i = 0; i < zarazka; i++)
    printf("%d%c", karty[i], (i == zarazka-1) ? '\n' : ' ');
}
```

Listing programu (Python)

```python
n, q = map(int, input().split())
karty = input().split()
kroky = input().split()

zarazka, reverz = 0, 1

for k in kroky:
    if k == 'D':
        zarazka -= reverz
    if k == 'H':
        zarazka += reverz
    if k == 'R':
        reverz = -reverz

if reverz == -1:
    karty = list(reversed(karty))
    zarazka = -zarazka

zarazka = ((zarazka % n) + n) % n

print(" ".join(karty[zarazka:] + karty[:zarazka]))
```