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

Vzorové riešenia 1. kola zimnej časti

vzorák napísal Prefix
(max. 15 b za riešenie)

1. Praktiky väznice Guantanámo

Táto úloha bola zameraná na to, aby ste sa naučili lepšie pracovať s permutáciami, ktoré určite využijete ako v programovaní, tak aj v matematike. **Permutácia** n **prvkov** je nejaké poradie n prvkov, také, že každý prvok využijeme práve raz. Dve permutácie sú rozdielne práve vtedy, keď majú aspoň na jednej pozícii rozdielne prvky. Napríklad $(1, 5, 2, 4, 3)$ je jednou z permutácií čísel $(1, 2, 3, 4, 5)$. V úlohe sme pracovali s permutáciami prvých písmen abecedy.

Podúloha a.

V prvej podúlohe ste mali zistiť, koľké v poradí je konkrétne meno v zozname – koľká v poradí je permutácia písmen "DBAC" v abecedne usporiadanom zozname všetkých permutácií. Keďže meno nie je až tak dlhé, všetkých možných permutácií tiež nebude až tak veľa a môžeme ich skúsiť všetky vypísať. Ako to však urobiť jednoducho?

Začneme permutáciami začínajúcimi na A a budeme pokračovať v abecednom poradí. Tak si budeme istí, že sa nám žiadna permutácia nezopakuje a že ich vypíšeme všetky.

Najprv si vypíšeme všetky permutácie začínajúce písmenom A. Pozrime sa na to, čo môže byť druhé písmeno. B, C alebo D (A to byť nemôže, lebo je už použité). Z nich je prvé v abecede B, takže vypíšeme všetky permutácie začínajúce dvojicou písmen AB. Existujú dve – "ABCD" a "ABDC". Ďalšie permutácie získame, až keď zmeníme druhé písmeno permutácie, čím dostaneme prvé dve písmená AC. Týmito písmenami začínajú permutácie "ACBD" a "ACDB". Dvojicou "AD" začínajú permutácie "ADBC" a "ADCB". Keď sa nám minú permutácie začínajúce na A, zmeníme prvé písmeno na B.

Možno ste si všimli, že v našom postupe si vyberieme prvé písmeno, a potom k nemu pripíšeme všetky permutácie zostávajúcich troch písmen. Keď sme si určili A ako prvé písmeno, zostali nám tri písmená – B, C, D. "BCD", "BDC", "CBD", "CDB", "DBC" aj "DCB" sú permutácie týchto troch písmen. Ak určíme ako prvé písmeno B, za neho napíšeme všetky permutácie trojice "ACD", atď.

Takýmto postupom vypíšeme všetky permutácie, a teda:

```
ABCD ABDC ACBD ACDB ADBC ADCB
BACD BADC BCAD BCDA BDAC BDCA
CABD CADB CBAD CBDA CDAB CDBA
DABC DACB DBAC DBCA DCAB DCBA
```

Tu si ľahko spočítame, že hľadaná permutácia (DBAC) je na 21. pozícii. Na túto podúlohu bola rozpisovacia metóda postačujúca, lebo nemáme veľa mien, iba 24.

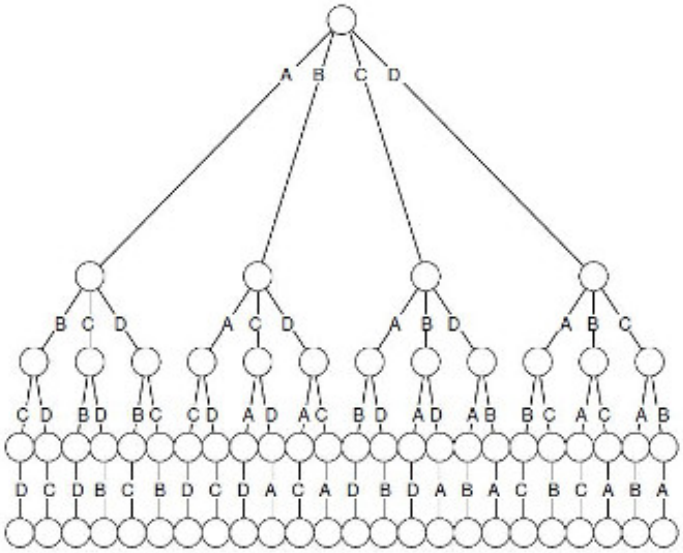
Rýchlejší postup by bol všimnúť si, že máme 6 permutácií začínajúcich písmenom A. Vieme, že permutácií začínajúcich na B aj na C je tiež 6 (rozmyslite si, prečo ich je rovnako veľa). Pred prvou permutáciou začínajúcou na D máme teda 18 permutácií začínajúcich A, B alebo C. Zostali nám už len permutácie začínajúce D-čkom a stačí nám teda už len zistiť poradie "DBAC" medzi nimi.

Podúloha b.

Pokračujeme druhou podúlohou. Chceme zistiť počet permutácií, ak máme 15 písmen. 15 je ale dosť veľké číslo, a tak to najprv skúsime s menším počtom – 4, rovnako ako v prvej podúlohe.

Skúsme si *vyrobiť* permutáciu. Na prvé miesto si môžeme zvoliť jedno zo 4 písmen – A, B, C, D. Na druhé miesto zostávajú 3 možnosti. Na tretie už len 2 a na posledné miesto dáme to jediné písmeno, ktoré nám zostalo. Situáciu pekne vidíme v strome permutácií.

V tomto strome máme úplne hore prvé miesto permutácie. Podľa toho, aké písmenko si vyberieme ako prvé, sa rozhodneme pre jeden zo štyroch menších stromov. Podľa toho, ktoré písmenko si vyberieme ako druhé, pokračujeme danou vetvou smerom dole. Na úplne spodnom poschodí stromu je 24 krúžkov – každý z nich



Obr. 1: Strom permutácií

predstavuje jednu permutáciu, ktorú zistíme podľa šípok, ktoré ku krúžku vedú. Takže napr. krúžok úplne napravo predstavuje permutáciu "DCBA", ten naľavo od neho zasa "DCAB". Krúžkov je dokopy 24 a teda máme 24 možných permutácií.

Tu si môžeme začať všímať, ako sa dá všeobecne počítať počet permutácií. Na prvé miesto vieme vybrať jeden zo štyroch prvkov, na druhé jeden z 3, na tretie už len z jeden dvoch a na posledné miesto nám zostal jediný prvok.

Na hornom poschodí stromu tak vychádzajú z krúžku 4 vetvy, na druhom poschodí vychádzajú z krúžkov 3 vetvy smerom dole, na ďalšom poschodí 2 vetvy z každého krúžku a na predposledom poschodí z krúžkov vychádza len 1 vetva.

Ak je počet krúžkov na spodnom poschodí rovnaký ako počet permutácií, stačí nám spočítať počet krúžkov. Tých je $4 \cdot 3 \cdot 2 \cdot 1 = 24$ a tak aj celkový počet permutácií je 24.

Keď chceme zistiť počet permutácií pre 15 písmen, môžeme si opäť skúsiť nakresliť takýto strom, lenže veľmi rýchlo narastie do enormných rozmerov. Je ale zrejmé, že na prvé miesto môžeme vybrať jedno z pätnástich písmen, na druhé už len jedno zo štrnástich, na tretie jedno z trinástich a tak ďalej. Takže to bude $15 \cdot 14 \cdot 13 \cdot 12 \cdot \cdots \cdot 3 \cdot 2 \cdot 1$.

Keďže sa táto operácia (vynásobebie všetkých čísel od 1 po n) využíva často, matematici jej dali názov – **faktoriál** – a označujú ju výkričníkom za číslom. Môžme povedať, že počet permutácií pätnástich písmen je faktoriál 15 alebo matematicky zapísané ako 15!. Všeobecne, počet permutácií n písmen je $n!$.

Na zisťovanie počtu permutácií sa dá pozrieť aj inak. Takýto pohľad sa obzvlášť zíde v ďalších podúlohách. Permutácie vytvárame tak, že si vyberieme jedno z písmen (napr. B) a dáme ho na prvé miesto. Potom za toto písmeno napíšeme všetky permutácie zvyšných písmen (A, C, D, E, ...). Takto dostaneme všetky permutácie začínajúce na toto písmeno (na písmeno B).

Označme si $p(15)$ počet permutácií 15 prvkov a predošlú úvahu vieme zapísať ako $p(15) = 15 \cdot p(14)$. Všeobecne, ak by sme si povedali že $p(n)$ bude počet permutácií n prvkov tak $p(n) = n \cdot p(n-1)$. Ak by sme chceli počítať počet permutácií 15 prvkov, môžeme podľa vzorca postupovať takto:

$$p(15) = 15 \cdot p(15-1) = 15 \cdot 14 \cdot p(14-1) = 15 \cdot 14 \cdot 13 \cdot p(13-1) = \cdots = 15 \cdot 14 \cdot 13 \cdot 12 \cdot \cdots \cdot 3 \cdot 2 \cdot p(2-1)$$

Keďže vieme, že permutácia jedného prvku je len jedna, tak $p(1) = 1$ a skutočne $p(15) = 15! = 1307674368000$.

Podúloha c.

Prejdime k tretej podúlohe, kde bolo treba zistiť koľko 5-prvkových permutácií začína písmenom D. Všimnime si, že akonáhle si určíme prvé písmeno, zostanú nám 4 voľné písmená (A, B, C a E) a 4 voľné pozície. Chceme

teda zistiť, koľko rôznych permutácií vieme vytvoriť z týchto 4 písmen. Použijeme znalosti z predchádzajúcej podúlohy a vieme, že to je práve $4! = 4 \cdot 3 \cdot 2 \cdot 1 = 24$.

Úvahu si môžeme aj zovšeobecniť. Ak máme permutáciu n písmen a nejakému z písmen určíme miesto, zostane nám $n-1$ písmen a $n-1$ miest. Potrebujeme teda vedieť počet permutácií $n-1$ prvkov, čo bude $(n-1)!$. Rozmyslite si, že rovnaký princíp by platil, ak by sme si stanovili nie prvé písmeno ale napr. tretie písmeno – teda že počet permutácií, kde je tretie písmeno B je rovnaký, ako počet permutácií, kde je prvé písmeno D.

Podúloha d.

V štvrtej úlohe ste mali zistiť poradie konkrétneho mena "DABCE" v zozname všetkých 5-písmenových permutácií. Vieme, že D je štvrté v abecede, takže pred "DABCE" musia byť v zozname permutácie začínajúce na A,B a C. V predošlej úlohe sme zistili, že n-písmenových permutácií začínajúcich na nejaké konkrétne písmeno je práve $(n-1)!$. Tu je $n = 5$, lebo máme dokopy 5 písmen. Takže počet permutácií začínajúcich na jedno z troch písmen A, B alebo C je $3 \cdot (5-1)! = 3 \cdot 4! = 3 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 72$. Pretože po prvom písmene nasledujú písmená "ABCE" a tieto písmena sú v abecednom poradí, "DABCE" je prvá permutácia začínajúca na D. Odpoveďou je teda 73.

Podúloha e.

Poznáme miesto permutácie "DABCE", potrebujeme ešte zistiť miesto permutácie "DEABC". Potom už bude ľahké spočítať, koľko premutácií je medzi nimi v zozname.

Môžeme použiť podobný postup, ako v minulej podúlohe. Opäť je prvé písmeno D, takže pred "DEABC" muselo byť aspoň 72 permutácií, ktoré nezačínali na D. Ďalej chceme zistiť pozíciu "DEABC" medzi permutáciami začínajúcimi na D. Zoberme si všetky písmená, ktoré môžu byť na druhej pozícii – A, B, C a E. Naša permutácia má na druhom mieste E, a teda pred ňou museli byť všetky permutácie začínajúce na DA, DB alebo DC.

A koľko pemutácií začína na dvoijice DA, DB alebo DC? Uvedomme si, že akonáhle už máme napísané písmená DA, zostávajú nám tri písmená, ktorými môžeme túto permutáciu dokončiť. Takže hľadáme permutácie troch písmen, ktorých je 3!. Na dvojice DA, DB alebo DC začína teda $3 \cdot 3! = 18$ permutácií.

Po E už nasledujú písmená v abecednom poradí, takže "DEABC" je prvá permutácia začínajúca písmenami DE. Celkovo je v zozname 72 permutácií začínajúcich na A, B, C a 18 permutácií začínajúcich na dvojice DA, DB, DC. Pred "DEABC" je teda $72 + 18 = 90$ permutácií, a preto je "DEABC" 91. permutácia.

Úloha znela: Koľko permutácií sa nachádza medzi "DABCE" a "DEABC". Teraz už vieme, že "DABCE" je v zozname 73. a "DEABC" je 91., takže medzi nimi sú permutácie s číslami $74, 75, 76...90$. To je dokopy $90 - 73 = 17$ permutácií.

Ak teda budeme vedieť rýchlo zistiť, koľká je daná permutácia, budeme vedieť aj rýchlo spočítať, koľko permutácií je medzi nejakými dvoma permutáciami $P1, P2$. Stačí spočítať poradie $P1$ a $P2$ a tieto poradia odčítať. Musíme si ale dať pozor, či chceme alebo nechceme započítať aj $P1, P2$.

Podúloha f.

V poslednej úlohe ste mali vymyslieť algoritmus, ktorý by vedel určiť poradie ľubovoľného mena v zozname. Ako sme už videli v predošlých úlohách, poradie permutácie zistíme tak, že spočítame počet permutácií pred ňou. Najprv si zoberieme dostatočne dlhý príklad na to, aby sa na ňom dal popísať postup algoritmu, napr. "FACDEB".

1. Vidíme, že prvé písmeno je F, a teda najprv zistíme, **koľko permutácií je v zozname pred ľubovoľnou permutáciou začínajúcou na F**. Sú to všetky, ktoré začínajú na A, B, C, D alebo E. Vieme, že takýchto permutácií je pre každé písmeno 5!, lebo permutujeme zvyšných 5 písmen. Dokopy muselo byť $5 \cdot 5! = 600$ permutácií, ktoré sú v zozname skôr ako tie, ktoré začínajú na F.

2. Ďalej chceme zistiť, koľká je permutácia "FACDEB" uprostred tých, čo začínajú na F. F-ko si teda môžeme pomyseľne vyškrtnúť, pretože všetky permutácie, s ktorými budeme ďalej pracovať, ho už majú na začiatku. Chceme teda zistiť, koľká je "ACDEB" medzi všetkými permutáciami písmen A, B, C, D, E. Opäť sa pýtame, **koľko permutácií je v zozname pred ľubovoľnou permutáciou začínajúcou na A**. V tomto kroku to máme uľahčené, pretože A je medzi A, B, C, D, E najskôr písmeno v abecede a teda pred "ACDEB" nie sú žiadne iné permutácie.

3. Vymažeme aj A a zostane nám "CDEB". C je medzi týmito písmenami druhé v abecede, takže pred "CDEB" museli byť permutácie 4 písmen začínajúce na B, a tých je $1 \cdot 3! = 6$, keďže permutujeme trojicu písmen B, C, E.

4. Zostala nám trojica písmen "DEB". D je medzi nimi druhé v abecede, a teda pred "DEB" sú permutácie 3 písmen začínajúce na B, ktorých je $1 \cdot 2! = 2$.

5. Keď vyškrtneme D, zostane nám "EB", a E je medzi nimi druhé v abecede takže pred "EB" sú permutácie 2 písmen začínajúce na B, a tých je $1! = 1$.

6. Nakoniec nám zostane len "B" a B je prvé v abecede medzi zostávajúcimi písmenami, a teda pred "B" nie sú žiadne permutácie.

Ako teda zistíme poradie "FACDEB" medzi všetkými permutáciami písmen A, B, C, D, E, F?

1. Najprv sme teda mali 600 permutácií, ktoré boli v zozname pred prvou permutáciou začínajúcou na F.
2. Potom sme mali 0 permutácií, ktoré sa začínali na F, ale druhé písmeno ešte nebolo A.
3. Na "FA" začínalo 6 permutácií predtým, než prišla prvá začínajúca na "FAC".
4. Potom nasledovali 2 permutácie, ktoré začínali "FAC" pred prvou permutáciou začínajucou na "FACD".
5. Ďalej 1 permutácia začínajúce na "FACD" pred "FACDE".
6. Na záver 0 permutácií začínajúcich na "FACDE", ktoré sú pred "FACDEB".

Pre určenie poradia "FACDEB" nám stačí spočítať všetky permutácie pred ňou. Stačí tak sčítať čísla, čo sme si doteraz zapísali: $5 \cdot 5! + 0 \cdot 4! + 1 \cdot 3! + 1 \cdot 2! + 1 \cdot 1! + 0 = 600 + 0 + 6 + 2 + 1 + 0 = 609$. Pred "FACDEB" je 609 permutácií, teda jej poradie je 610.

Teraz bude jednoduché zapísať všeobecný postup. Označíme si X permutáciu, ktorej poradie hľadáme:

```
Ako zistiť, koľko permutácií bolo pred permutáciou X:
    Výsledok je zatiaľ 0.
    Opakuj, kým je v X aspoň 1 písmeno:
        Zistíme počet písmen v X, a označíme si ho n
        Zistíme počet písmen v X, ktoré sú v abecede pred prvým písmenom X, a označíme si ho p
        K výsledku pripočítame p*(n-1)!
        Odstránime z X prvé písmeno
```

Keďže permutácia X bola konečne dlhá a vždy z nej odstraňujeme jedno písmeno, po nejakom čase sa určite dostaneme do momentu, kedy nám nezostane žiadne písmeno. Vieme tak, že algoritmus vždy skončí.

Druhou podmienkou, ktorú požadujeme od algoritmu je, aby vypočítal správne riešenie. V úlohe **c.** sme si ukázali, že permutácií n písmen, kde je pevne určené prvé písmeno je $(n-1)!$. Algoritmom sme sčítali všetky permutácie pred danou permutáciou: v prvom opakovaní všetky tie, ktoré sa líšia v prvom písmene, v druhom opakovaní tie ktoré majú rovnaké prvé písmeno ale iné druhé, atď. Započítali sme teda každú permutáciu pred X práve raz.

Pokiaľ ste prišli na algoritmus, bolo ho ešte potrebné využiť na zistenie poradia niekoľkých permutácií. Toto už bola jednoduchá práca s kalkulačkou, len bolo potrebné niekoľkokrát zopakovať tento postup.

Poradia permutácií sú nasledovné, no za numerické chyby body nestratíte.

FACDEB – 610, AEGFDBCH – 2725, DIECAHFGB – 158662, BHACDEFGJI – 604802

Tí, čo vedia programovať si mohli uľahčiť prácu oproti ručnému vypisovaniu a počítaniu faktoriálov na kalkulačke.

Listing programu (C++)

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

int Faktorial[20];

int main() {
    Faktorial[0]=1;
    for(int i=1; i<20; i++) Faktorial[i] = i*Faktorial[i-1];
    string s;
    cin >> s;
    int vysledok=0;
    int n=s.length();
    for(int i=0; i<n; i++) {
        int mensie=0;
        for(int j=i; j<n; j++)
            if(s[j]<s[i]) mensie++;
        vysledok += mensie*Faktorial[n-i-1];
    }
    cout << vysledok << endl;
}
```

Listing programu (Python)

```
from math import factorial
X = input()

vysledok = 0

while len(X) > 0:
        n = len(X)

        p = 0
        for pismeno in X:
                if pismeno < X[0]:
                        p += 1

        vysledok += p * factorial(n-1)
        X = X[1:]

print(vysledok)
```

2. Praktické regulárne výrazy

vzorák napísal Žaba
(max. 15 b za riešenie)

Podúloha a.

Vytvoriť telefónne číslo nie je ťažké. Stačí postupne naskladať všetko, čo bolo popísané v zadaní. Vieme, že čísla začínajú kombináciou „09", túto časť teda môžeme nechať tak, ako je. Nasleduje dvojčíslie udávajúce operátora. Jedna možnosť je pomocou „|" poskladať všetky možnosti. Vyzeralo by to takto: „05|06|07|08|15|16|17|18". Môžeme si to však skrátiť, ak si všimneme, že tieto čísla začínajú buď na „0" alebo „1" a pokračujú jednou z cifier „5", „6", „7" a „8". A ľubovoľné takto vzniknuté číslo je dobré. Dostávame teda regulárny výraz „(0|1)[5678]".

Nasleduje šesť ľubovoľných cifier. Jednu ľubovoľnú cifru spravíme jednoducho pomocou „[0-9]". Opakovať však vieme iba neurčitý počet krát. Ak chceme niečo zopakovať práve šesť krát, nezostáva nám nič iné, ako tento vzor nalepiť šesťkrát za sebou.

Výsledný regulárny výraz, ktorý akceptuje telefónne čísla je:
„09(0|1)[5678][0-9][0-9][0-9][0-9][0-9][0-9]".

Podúloha b.

Táto úloha je ešte jednoduchšia. Stačí si uvedomiť, že ak máme reťazec zložený iba z písmen „a" a tento počet je deliteľný tromi, tak tieto a-čka sa dajú rozdeliť do trojíc. Keď si teda zoberieme regulárny výraz, ktorý vyzerá ako „(aaa)*", dostaneme práve to, čo potrebujeme. Ak hviezdička zopakuje daný reťazec k-krát, výsledný reťazec bude mať $3k$ a-čok. A hviezdička môže zopakovať reťazec ľubovoľne, aj nulakrát.

Podúloha c.

Máme vytvoriť regulárny výraz, ktorý akceptuje reťazce nepárnych dĺžok, ktoré sa skladajú iba z písmena „a". Keby bola úloha vytvoriť párne dlhé reťazce, použili by sme riešenie podobné podúlohe b. Dostali by sme regulárny výraz „(aa)*". No a získať z párnych reťazcov nepárne je jednoduché. Pridáme na koniec ešte jedno „a". Takže hľadaný regulárny výraz je „(aa)*a".

Podúloha d.

V tejto podúlohe bolo najdôležitejšie nezabudnúť na nejaké špeciálne prípady. Napríklad, reťazec zo samých „c" spĺňa podmienku zo zadania. Takisto reťazec, v ktorom sa nevyskytuje písmeno „b" je korektný. Je dobrým zvykom pozrieť sa na tieto špeciálne prípady a keď úlohu vyriešime, overiť si, že náš regulárny výraz funguje aj pre ne.

Každý výsledný reťazec si vieme rozdeliť na dve časti. Začiatočná časť, v ktorej sa nevyskytuje písmeno „b" a koncová časť, v ktorej sa nevyskytuje písmeno „a". Toto rozdelenie vyplýva z toho, že žiadne „b" nie je pred žiadnym „a".

Vytvoriť regulárny výraz pre prvú časť je jednoduché. Je to ľubovoľný reťazec, v ktorom sa nachádzajú iba písmená „a" a „c". Všetky takéto reťazce môžeme generovať tak, že začneme prázdnym slovom a postupne na koniec priliepame jedno z povolených písmen. A toto môžeme pomocou regulárnych výrazov zapísať ako „[ac]*" (rovnako dobré môže byť použitie „(a|c)*").

Rovnakým postupom vieme druhú časť generovať pomocou „[bc]*". Takže vo výslednom regulárnom výraze nám stačí tieto dva reťazce zreťaziť a dostaneme: „[ac]*[bc]*". Vidíme, že to naozaj generuje výrazy, kde sú všetky „a" pred „b". Teraz sa pozrime, či to spĺňa aj všetky špeciálne prípady, ktoré sme si spomenuli na

začiatku. Vieme dostať reťazce tvorené iba písmenami „c". Proste sa nikdy nevyberie žiadne iné písmeno. Reťazce neobsahujúce „b" vzniknú tak, že druhá hviezdička sa nezopakuje ani raz. A naopak to platí pre reťazce neobsahujúce „a".

Podúloha e.

Táto podúloha bola samozrejme najťažšia. Postupne sa preto pustíme do jej riešenia. Keby sme mali úlohu, kde môžeme použiť iba písmená „a" a „b" a žiadne dve za sebou idúce písmená nemôžu byť rovnaké, úloha by sa výrazne zjednodušila. Tieto písmená by totiž museli ísť nastriedačku za sebou. To vieme dosiahnuť napríklad pomocou „(ab)*". Problém ale je, že toto nepokrýva všetky možnosti. Všetky tieto reťazce totiž začínajú na „a" a končia na „b". My však potrebujeme rátať aj s takými, ktoré začínajú na „b" alebo končia na „a". Na začiatok teda *možno* pridáme „a" a na koniec *možno* pridáme „b". Na niečo také nám predsa slúži znak „?". Dostaneme teda regulárny výraz „b?(ab)*a?".

Teraz však do toho potrebujeme pridať ešte písmená „c". Zoberme si ľubovoľný platný reťazec, ktorý môže obsahovať všetky tri písmená a žiadne dve za sebou idúce písmená nie sú rovnaké. Pozrime sa teraz na všetky výskyty písmena „c". Uvedomme si, že medzi dvoma za sebou idúcimi c-čkami v reťazci je **neprázdny** reťazec, ktorý obsahuje iba písmená „a" a „b" a žiadne dve za sebou idúce písmená nie sú rovnaké. A to je predsa úloha, ktorú sme už riešili v predchádzajúcom odseku.

Teda takmer. V predchádzajúcom riešení sme dovoľovali, aby vzniknutý reťazec bol aj prázdny, v tomto prípade to však tak nemôže byť. Upravme teda regulárny výraz „b?(ab)*a?" tak, aby akceptoval iba neprázdne reťazce z „a" a „b", kde žiadne dve za sebou idúce písmená nie sú rovnaké. Ak chceme, aby boli tieto reťazce neprázdne, musíme zaručiť, že sa vytvorí aspoň jeden znak. Napríklad ten prvý. Ak by tento reťazec začínal na písmeno „b", môže za ním nasledovať ľubovoľne veľa (aj nula) dvojíc „ab" a následne sa možno skončí na „a". Takému niečomu zodpovedá regulárny výraz „a(ba)*b?". No a ak tento reťazec začína na „b", tak za ním nasleduje ľubovoľne veľa dvojíc „ab" a na konci je možno ešte „a". Teda máme regulárny výraz „b(ab)*a?". A aby som generoval všetky reťazce, ktorým vyhovuje prvý alebo druhý spomenutý regulárny výraz, spojím ich pomocou „|". Výsledný výraz bude teda „(a(ba)*b?)|(b(ab)*a?)".

Práve vytvorený regulárny výraz budeme často potrebovať v našom ďalšom postupe. Označme si ho teda veľkým písmenom „F". Keď teraz v regulárnom výraze napíšem „F", znamená to, že tam chcem vložiť celý výraz „(a(ba)*b?)|(b(ab)*a?)". Ako teraz vyzerajú všetky reťazce prvých troch písmen abecedy, kde sa žiadne dve za sebou idúce neopakujú?

Tak ako v predchádzajúcom riešení, rozoberieme si dve možnosti. Môžeme začať písmenom „c". Po ňom bude nasledovať nejaká dobrá postupnosť a-čok a b-čok, opäť „c" atď, až kým to neskončí buď na tom c-čku alebo ešte jednou postupnosťou a-čok a b-čok. Toto zapíšeme ako „c(Fc)*F?". Alebo začnem nejakými a-čkami alebo b-čkami, potom pôjde „c", opäť „a" a „b" atď, a na koniec sa možno ešte pridá jedno „c". To nám dá regulárny výraz „F(cF)*c?". Tým sme rozobrali všetky možnosti ako môže dané slovo vyzerať. Jediné čo nám uniká je prázdny reťazec. Ten však vieme pridať veľmi jednoducho tak, že na koniec pridáme ešte jednu „|", za ktorou už nebude nič – teda prázdny reťazec. Dostaneme „(c(Fc)*F?)|(F(cF)*c?)|".

A ak to rozpíšeme poriadne (a každé „F" ešte dáme do zátvorky a pre prehľadnosť to dáme do dvoch riadkov):

```
"(c(((a(ba)*b?)|(b(ab)*a?))c)*((a(ba)*b?)|(b(ab)*a?))?)|
(((a(ba)*b?)|(b(ab)*a?))(c((a(ba)*b?)|(b(ab)*a?)))*c?)|"
```

Nepôsobí to síce najkrajšie a dajú sa vymyslieť aj mierne jednoduchšie výrazy[1], napríklad: „b?(ab)*a?(c((a(ba)*b?)|(b(ab)*a?)))*c?", ale je pomerne jednoduché naň prísť.

vzorák napísal Andrej a Roman
(max. 15 b za riešenie)

3. Prísne tajné správy

Pri riešení tejto úlohy bol potrebný nejaký obrázkový editor (program na prácu s obrázkami). V tomto vzorovom riešení si ukážeme, ako sa dala riešiť pomocou editorov GIMP[2] a Skicár[3]. Samozrejme, väčšina vecí by sa v iných editoroch dala riešiť veľmi podobne.

Podúloha 1

Ako prvé si môžeme všimnúť, že v zadaní sa nám vyskytuje niečo, čo sa podobá na QR kód [4]. Skúsime

[1] Môžete sa zamyslieť, ako vznikol tento konkrétny. Všimnite si, že obsahuje časti spomínané v tomto vzorovom riešení, akurát ich naskladá šikovnejšie.
[2] http://www.gimp.org/
[3] https://sk.wikipedia.org/wiki/Skic%C3%A1r_(softv%C3%A9r)
[4] viac o QR kódoch nájdete na https://sk.wikipedia.org/wiki/QR_k%C3%B3d

teda nájsť nejaký nástroj na prečítanie tohto QR kódu. Môžeme použiť napríklad Webqr [5]. Keď však obrázok načítame, zistíme, že kód je neplatný. Ako sa náš kód líši od iných, resp. čo ho robí neplatným? Sú to vymenené (invertované) farby. Čiernu nahradila biela a naopak. Keď chceme získať pôvodný kód, musíme farby vymeniť naspäť. V Skicári to urobíme nasledovne: po otvorení obrázka si ho označíme a po kliknutí pravým tlačidlom sa nám ukáže možnosť invertovať farbu. V GIMP-e je treba po otvorení obrázka použiť `Farby->Invertovanie hodnôt`. Po tomto zásahu nám už nástroj na rozpoznávanie QR kódov vypíše správny link:
`https://prask.ksp.sk/specialne/prask/2/1/3/1/uplne_najtajnejsi_link_ktory_nikdy_nijaky_fiskus_neuhadne`

Podúloha 2

V tejto podúlohe išlo o to, že náš hľadaný link je napísaný úplne naspodku, farbou, ktorú nemôže ľudské oko rozoznať od pozadia. To znamená, že na prečítanie musíme tento text od pozadia nejakým spôsobom odlíšiť. Ak pozorne prečítame viditeľnú časť postupne sa strácajúceho textu, mali by sme si všimnúť, že nás nabáda hneď k dvom riešeniam úlohy. Prvé, jednoduchšie, je otvoriť si obrázok v Skicári a použiť nástroj plechovka na zafarbenie pozadia inou farbou ako čiernou alebo bielou, potom je už link v obrázku jasne viditeľný. Druhé riešenie, o čosi zložitejšie, je otvoriť si program v GIMP-e a v časti `Farby`, nájsť nastavenia jasu a kontrastu a obe tieto hodnoty zvýšiť na maximum. Link, ktorý bol riešením:
`https://prask.ksp.sk/ucet/login/?next=/specialne/prask/2/1/3/2/heslo_je_afmnskm/`

Podúloha 3

Keď sa lepšie pozrieme na obrázok, môžeme si všimnúť, že je to len veľmi natiahnutý text. Obrázok je v tomto stave nečitateľný, preto ho musíme zmeniť na čitateľnú veľkosť. Buď tak, že ho roztiahneme do šírky alebo mu zmenšíme výšku.

Riešenie v skicári: treba roztiahnuť papier na veľkosť približne 500×20000 pixelov (výška $\times$ šírka), prípadne ho zmenšiť na zhruba 15×500 pixelov. Musíme preto použiť nástroj `Zmeniť veľkosť` v záložke `Domov`, kde nastavíme požadované rozmery. Pozor si treba dať na možnosť `Zachovať pomer strán`, ktorú musíme vypnúť.

Riešenie v GIMP-e: Podobne ako pri Skicári iba zmeníme rozmery obrázka nástrojom `Zmeniť mierku` (v anglickej verzii `Scale image`), ktorý nájdeme v záložke `Obrázok` (`Image`). Nastavíme rozmery obrázka, pričom si znovu musíme dať pozor na zachovanie pomeru strán. Výsledný odkaz:
`https://prask.ksp.sk/specialne/prask/2/1/3/3/je_naozaj_tazke_vymysliet_kreativny_tazko_hadatelny_link`

Podúloha 4

V tejto podúlohe máme až dva obrázky, pričom oba na prvý pohľad vyzerajú ako náhodný čiernobiely šum. Môžeme si všimnúť, že oba obrázky majú rovnakú veľkosť v pixeloch. To by nám mohlo napovedať, že obrázky treba prekryť. V GIMP-e sa to dá urobiť tak, že obrázky otvoríme ako dve vrstvy jedného obrázka. To môžeme urobiť napríklad tak, že najprv otvoríme jeden obrázok a potom pomocou možnosti `Súbor->Otvoriť ako vrstvu` otvoríme druhý.

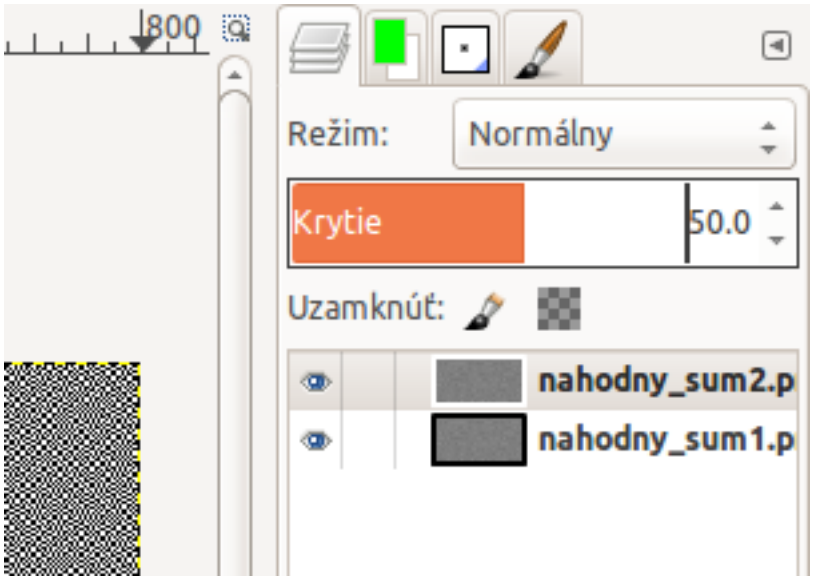
Vrstvy v GIMP-e si môžete predstaviť ako papiere, ktoré sú položené na sebe. Niektoré papiere môžu byť priesvitné alebo čiastočne priesvitné. Vtedy je vidieť papier, ktorý je pod nimi. V našom prípade sú obe vrstvy nepriesvitné, preto vidíme iba vrchnú (druhý obrázok). V dialógovom okne `Vrstvy` (v anglickej verzii `Layers`)[6] môžeme vidieť dva riadky s malými ikonkami našich dvoch vrstiev a ich názvami (názvy pôvodných obrázkov), pričom jedna z nich by mala byť označená. Ak je označená spodná vrstva, klikneme na vrchnú, čím ju označíme. Potom jej zmenšíme `Krytie` (`Opacity`) na `50 %`. Tým sa vrstva stane polopriesvitnou a uvidíme čiastočne aj vrstvu pod ňou (prvý obrázok). Teraz už dokážeme prečítať link, ktorý je sivý, kým zvyšok pozadia je čiernobiely šum.

Výsledný link je:
`https://prask.ksp.sk/specialne/prask/2/1/3/4/poculi_ste_o_spravnom_konovi_baterie_zosivajucom/`

Podúloha 5

Táto úloha mala dve riešenia, "poctivé" a "nepoctivé". To poctivé bolo pozerať GIF stále dookola a nejakým spôsobom vyčítať link. Vzorové riešenie bolo otvoriť animáciu v nejakom programe na to určenom. Túto konkrétnu úlohu s animáciou zvláda aj GIMP a nie je potrebný nejaký špeciálny program zameraný na animácie. Animácia je vlastne postupnosť po sebe idúcich obrázkov. GIMP nám tieto obrázky otvorí ako jednotlivé vrstvy jedného obrázka, pričom prvý obrázok animácie bude spodná vrstva a posledný obrázok bude vrchná vrstva. Po otvorení teda uvidíme posledný obrázok animácie, ktorý je celý čierny. Ten zneviditeľníme (buď tak,

[5] https://webqr.com/index.html
[6] ak ho nemáte otvorené, tak kliknite na `Okná->Dokovateľné dialógy->Vrstvy` (`Windows->Dockable dialogs->Layers`)



že mu znížime krytie na nulu alebo klikneme na ikonku oka vedľa tohto obrázka v okne `Vrstvy`) a uvidíme predposledný obrázok, na ktorom je napísaný link:

`https://prask.ksp.sk/specialne/prask/2/1/3/5/dufam_ze_ste_tuto_podulohu_vyriesili_poctivo`

Podúloha 6

Tento obrázok má iba jeden čierny pixel a nemá sám o sebe žiadnu výpovednú hodnotu. Preto ho ani nemá zmysel nejak graficky upravovať. Samotný obrázok ale okrem primárnych údajov (informácie v pixeloch) nesie aj sekundárne údaje (metadáta), ktoré môžu hovoriť o obrázku niečo viac (napríklad autora, v prípade fotiek dátum a čas odfotenia, aké má mať obrázok rozmery pri tlači atď.). Skúsme tieto metadáta preskúmať. Na internete je kopa nástrojov na prehľadávanie metadát. Jeden z nich je na `http://regex.info/exif.cgi`. Keď sa nám zobrazia metadáta, môžeme spozorovať, že komentár k tomuto obrázku obsahuje náš hľadaný link:

`https://prask.ksp.sk/specialne/prask/2/1/3/6/tam_kde_nic_nie_je_sa_ani_nic_nemoze_stratit/`

Iným spôsobom riešenia tejto úlohy bolo otvoriť si obrázok v obyčajnom textovom editore (t. j. napríklad Notepad, Gedit, Vim, ...). Uvidíme hromadu nezmyselných znakov a niekde medzi nimi úsek, ktorý dáva zmysel: náš link.

4. Pevná veža z kociek

vzorák napísal Kubo
(max. 15 b za riešenie)

Aj napriek tomu, že táto úloha nebola myšlienkovo náročná, dala sa riešiť jednoduchším a komplikovanejším spôsobom. Častokrát, práve to komplikovanejšie riešenie je to, čo nás napadne ako prvé, oplatí sa však nad úlohou trochu zamyslieť a prísť s riešením, ktoré je lepšie. To nám totiž ušetrí kopec problémov a chýb.

Najjednoduchšie čo nás napadne je postupne pre každú kocku vo for-cykle načítať viditeľné čísla, z čoho vieme zistiť, ktoré čísla sú skryté. Tie následne sčítame dokopy a máme riešenie.

Môžeme si však uvedomiť, že nepotrebujeme vedieť ktoré čísla sú zakryté. Zaujíma nás totiž iba ich súčet. Keďže vieme, že každá kocka má rovnaký súčet čísel na jej stenách ($1+2+3+4+5+6=21$), tak nám stačí zrátať si čísla, ktoré vidíme a odrátať ich od súčtu čísel na kocke. Celé riešenie teda vyzerá tak, že pre každú kocku načítame viditeľné strany, odrátame ich od 21 a tieto čiastkové výsledky spočítame.

Rovnako dobré a možno o malý kúsok ľahšie riešenie na implementáciu je najskôr si zrátať súčet čísel na celej veži (počet kociek $\times$ súčet čísel na jednej kocke) a od neho postupne odrátať všetky čísla, ktoré sú uvedené na vstupe, keďže to sú čísla, ktoré vidíme.

Listing programu (C++)

```
#include <iostream>

using namespace std;

int main(){

    int n;
    cin >> n;
```

```
    int vsetky = 21*n;   // zratame si celkovy sucet cisel na kockach

    int vidim;
    cin >> vidim;    // nacitame hornu stranu prvej kocky
    vsetky -= vidim; // a odratame od celkoveho suctu

    // pre kazdu kocku chceme nacitat viditelne cisla
    for (int i=0; i<n; i++){
        for (int j=0; j<4; j++){
            cin >> vidim;    // postupne nacitame styri cisla z toho isteho riadku
            vsetky -= vidim; // a odratame ich od celkoveho suctu
        }
    }

    cout << vsetky << endl;

    return 0;
}
```

Listing programu (Pascal)

```
Program kocky;

var
  n, vsetky, kocka, i, j: Integer;

begin

  Readln(n);
  vsetky := 21*n;  // zratame si celkovy sucet cisel na kockach

  Readln(kocka);            // nacitame hornu stranu prvej kocky
  vsetky := vsetky - kocka; // a odratame od celkoveho suctu

  // pre kazdu kocku chceme nacitat viditelne cisla
  for i := 1 to n do
    for j := 1 to 4 do
      begin
        Read(kocka); // postupne nacitame styri cisla z toho isteho riadku
        vsetky := vsetky-kocka; // a odratame ich od celkoveho suctu
      end;

  Writeln(vsetky);

end.
```

Listing programu (Python)

```
n = int(input())

# vyratame si celkovy pocet kociek
vsetky = 21*n

vrchna = int(input())
vsetky -= vrchna;

for i in range(n):
    # nacitame riadok a rozdelime ho podla medzier na styri casti
    vidim = input().split()

    for x in vidim:
        # postupne odratame jednotlive cisla
        vsetky -= int(x)


print(vsetky)
```

vzorák napísal Kuko
(max. 15 b za riešenie)

5. Perfektné leukoplasty

Ak by sme našli obsah prieniku obdĺžnikov[7], riešenie by bolo jednoduché. Sčítali by sme obsahy obdĺžnikov a odrátali obsah ich prieniku, keďže ten sme zarátali dvakrát.

Ako na obsahy obdĺžnikov?

V prvom rade musíme vedieť, ako zrátať obsahy našich obdĺžnikov. Tie vypočítame ako súčin šírky a výšky, ktoré vieme spočítať rozdielom súradníc. Výšku vyrátame ako súradnicu hornej strany mínus súradnicu dolnej, šírku ako súradnicu pravej strany mínus súradnicu ľavej).

Už teda vieme, ako zistíme zo súradníc obdĺžnika jeho obsah. No aj prienik našich obdĺžnikov je len obdĺžnik. Zrátať ho teda môžeme rovnako, stačí len zistiť jeho súradnice.

[7] *Prienik* je iba vznešenejší názov pre tú oblasť, kde sa leukoplasty prekrývajú

Ach, prienik, kde si?

Jednou možnosťou bolo začať veľa kresliť a postupne nájsť všetky možnosti, ako môže prienik vyzerať – prienik neexistuje, prenikajú sa rohmi, stranami, jeden obdĺžnik je celý vo vnútri toho druhého... Rozlíšiť medzi týmito možnosťami sa síce dalo pomocou komplikovaných if-ov, bolo však ťažké nezabudnúť nejakú možnosť. Pokúsime sa teda nájsť riešenie, ktoré je oveľa všeobecnejšie.

Keby sme porovnali ľavé strany obdĺžnikov, tak vieme povedať, že prienik nebude ohraničený tou viac naľavo. To znamená, že určite bude napravo od oboch ľavých strán. Ak teda prienik existuje, tak jeho ľavá strana je na rovnakej súradnici ako pravšia z ľavých strán štvorcov, teda tá, čo má väčšiu x súradnicu.

Rovnakú úvahu môžeme zopakovať pre dolné strany – dolná strana prieniku je na rovnakej súradnici ako hornejšia z dolných strán štvorcov, teda tá, čo má väčšiu y súradnicu.

Pre horné a pravé strany vieme spraviť niečo podobné, akurát zrkadlovo otočené. Horná a pravá strana nášho prieniku musí byť na súradnici nižšej hornej strany (menšia y súradnica) a ľavšej pravej strany (menšia x súradnica) našich štvorcov.

Je tu ale malý problém – čo ak obdĺžniky nemajú prienik? V podmienkach predsa rátame s tým, že prienik existuje. V takom prípade sa nám stane akurát to, že výška alebo šírka prieniku nám vyjde menšia alebo rovná nule. Avšak toto si vieme skontrolovať a v takomto prípade vieme, že obsah prieniku je nula.

Na výstup už iba vypíšeme obsah jedného obdĺžnika plus obsah druhého obdĺžnika mínus obsah prieniku.

Listing programu (C++)

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

int main() {
    //suradnice obdlznikov
    int x11,x12,x21,x22,y11,y12,y21,y22;
    cin >> x11 >> y11 >> x12 >> y12;
    cin >> x21 >> y21 >> x22 >> y22;
    //suradnice prieniku
    int px1,px2,py1,py2;
    px1 = max(x11,x21); py1 = max(y11,y21);
    px2 = min(x12,x22); py2 = min(y12,y22);
    int prienik;
    if(px2-px1 <= 0 || py2-py1 <= 0) prienik=0;
    else prienik = (px2-px1)*(py2-py1);
    int obsah1 = (x12-x11)*(y12-y11),obsah2 = (x22-x21)*(y22-y21);
    cout << obsah1 + obsah2 - prienik << endl;
}
```

Listing programu (Pascal)

```
program Hello;
var
//na obdlzniky 1 a 2
vpravo1, vpravo2, vlavo1, vlavo2, hore1, hore2, dole1, dole2: longint;

//na prienik
vpravoP, vlavoP, horeP, doleP: longint;

//pomocne premmene na vysku - sirku
vyska1, sirka1, vyska2, sirka2, vyskaP, sirkaP: longint;

//obsahy
obsah1, obsah2, obsahP: longint;

begin
    //nacitame vstup - ohranicenia oblznikou
    readln(dole1, vlavo1, hore1, vpravo1);
    readln(dole2, vlavo2, hore2, vpravo2);

    //pocitame mozne strany prieniku
    if vlavo1 < vlavo2 then vlavoP := vlavo2
    else vlavoP := vlavo1;

    if dole1 < dole2 then doleP := dole2
    else doleP := dole1;

    if vpravo1 > vpravo2 then vpravoP := vpravo2
    else vpravoP := vpravo1;

    if hore1 > hore2 then horeP := hore2
    else horeP := hore1;

    //pocitanie obsahu prieniku
    vyskaP := horeP - doleP;
    sirkaP := vpravoP - vlavoP;

    obsahP := vyskaP * sirkaP;
    if (vyskaP <= 0) or (sirkaP <= 0) then obsahP := 0;
```

```
    //pocitanie obsahu velky obdlznikou
    vyska1 := hore1 - dole1;
    sirka1 := vpravo1 - vlavo1;

    obsah1 := vyska1 * sirka1;

    vyska2 := hore2 - dole2;
    sirka2 := vpravo2 - vlavo2;

    obsah2 := vyska2 * sirka2;

    writeln(obsah1 + obsah2 - obsahP);
end.
```