PRASK
III. ročník, 2016/17
Katedra základov a vyučovania informatiky FMFI UK,
Mlynská Dolina, 842 48 Bratislava

Vzorové riešenia 1. kola zimnej časti

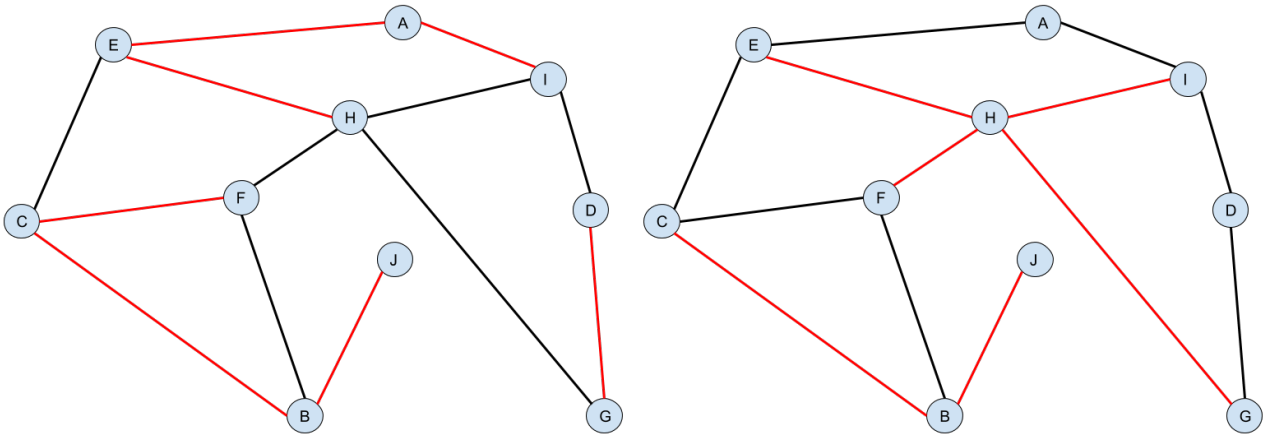
vzorák napísal Žaba
(max. 15 b za riešenie)

1. Postav cesty!

Na začiatok si pripomeňme zadanie úlohy. Dostali sme mapu Absurdistanu, na ktorej bolo n miest, medzi ktorými viedli nejaké cesty. Našou úlohou bolo vybrať cesty, ktoré sa budú rekonštruovať. Cieľom bolo vybrať takýchto ciest čo najmenej, lebo rekonštrukcia je drahá, ale zároveň docieliť, aby sa medzi každými dvoma mestami dalo cestovať iba po zrenovovaných cestách.

Skôr ako sa pustíme do riešenia, musíme si zadefinovať pojem **komponent**, ktorý budeme počas tohto vzorového riešenia hojne používať. Komponentom nazývame skupinu miest, medzi ktorými sa dá pohybovať iba po zrekonštruovaných cestách.

Na obrázku nižšie si môžete pozrieť ako môže vyzerať Absurdistan. Červenou farbou sú označené zrekonštruované cesty. Na obrázku naľavo sú 3 komponenty – (`A`, `E`, `H`, `I`), (`D`, `G`) a (`C`, `B`, `F`, `J`). Na obrázku napravo sú komponenty až 4 – (`A`), (`E`, `H`, `I`, `F`, `G`), (`D`) a (`C`, `B`, `J`). Všimnite si, že aj osamotený vrchol tvorí komponent.



Ak sme ešte nezrekonštruovali žiadnu cestu, v Absurdistane sa nachádza n komponentov – každé mesto tvorí samostatný komponent odkiaľ sa nevieme pohnúť ďalej. No a našou úlohou je zrekonštruovať také cesty, aby bol v Absurdistane iba jeden komponent, ktorý obsahuje všetky mestá. Potom sa z definície dá dostať z každého mesta do každého iného iba po zrekonštruovaných cestách.

Podúlohy a) a b)

Ako však zistíme, ktoré cesty chceme vybrať? Skúsme ich postupne rekonštruovať, je úplne jedno v akom poradí. Aby sme však nezrenovovali všetky cesty, vždy pred renováciou si položíme dôležitú otázku: Pomôže nám rekonštrukcia tejto cesty?

Predstavme si, že uvažujeme o ceste, ktorá vedie medzi mestami `X` a `Y`. V úvahu pripadajú dve možnosti. Obe tieto mestá ležia v tom istom komponente, teda už sa medzi nimi dá pohybovať iba po zrekonštruovaných cestách. V takom prípade nám však rekonštrukcia cesty medzi `X` a `Y` v ničom nepomôže. Ak už existoval spôsob ako medzi týmito dvoma mestami prejsť iba po zrekonštruovaných cestách, vždy, keď by sme chceli použiť túto priamu cestu, môžeme to radšej obísť. A ušetríme si jednu rekonštrukciu.

Druhá možnosť nastane ak mestá `X` a `Y` neležia v tom istom komponente. V takom prípade nám rekonštrukcia tejto cesty pomôže, lebo odteraz sa budeme vedieť dostať z `X` do `Y` (a naopak). A nielen to. Ak mesto `A` bolo v tom istom komponente ako `X` a mesto `B` v tom istom komponente ako `Y`, tak zrazu sa vieme dostať aj z mesta `A` do mesta `B` a to iba po zrekonštruovaných cestách. Stačí ak najskôr pôjdeme z `A` do `X` (takýto spôsob existuje, lebo sú v tom istom komponente), potom prejdeme po novozrekonštruovanej ceste do `Y` a odtiaľ do mesta `B` (takýto spôsob existuje, lebo sú v tom istom komponente). Zrekonštruovanie tejto cesty nám teda spojilo príslušné dva komponenty a nahradilo ich jedným väčším.

A to je v konečnom dôsledku náš cieľ. Spojiť n samostatných komponentov do jedného veľkého. Takýto postup vieme aj veľmi jednoducho zapísať a pochopia ho aj vaši rodičia:

```
postupne pre každú cestu v Absurdistane sprav:
    ak sa mestá na koncoch tejto cesty nachádzajú v rovnakom komponente
    (dá sa medzi nimi dostať iba po zrekonštruovaných cestách), tak:
        s touto cestou nič nerob
    ak sa nenachádzajú v rovnakom komponente, tak:
        zrekonštruuj túto cestu
```

Tento postup určite povedie k tomu, že nám ostane jeden veľký komponent a teda sa budeme vedieť dostať odvšadiaľ všade iba po zrekonštruovaných cestách. Vyberie však tento postup minimálny počet ciest? A koľko ich vlastne vyberie?

Spomeňme si, čo sa stane ak v tomto postupe zrekonštruujeme nejakú cestu – spojíme dva komponenty do jedného väčšieho. To znamená, že počet komponentov klesne o 1. No a ak sme začínali s n komponentami (každé mesto je samostatný komponent) a skončili sme s 1 komponentom, museli sme takéto spájanie spraviť presne $n-1$ krát. **Zrekonštruovali sme teda $n-1$ ciest**.

Toto je však aj dôvod, prečo je tento počet zrekonštruovaných ciest najmenší možný. Ľubovoľná rekonštrukcia niektorej cesty vie spojiť najviac dva komponenty. A preto musíme zrekonštruovať aspoň $n-1$ ciest.

Podúloha c)

V podúlohe c) je situácia trochu komplikovanejšia. Za rekonštrukciu cesty totiž musíme zaplatiť, v tomto prípade 1 alebo 2 milióny dolárov. Našou úlohou je opäť zrekonštruovať také cesty, aby sa medzi každými dvoma mestami dalo pohybovať iba po zrekonštruovaných cestách, ale zároveň chceme zaplatiť čo najmenej peňazí.

Pri predchádzajúcej úlohe sme si ukázali, že musíme postaviť aspoň $n-1$ ciest. Táto podmienka sa nezmení ani teraz. Ak by sme totiž postavili viac ciest, určite sa nájde taká, ktorej nezrekonštruovanie neporuší podmienku zo zadania. A ak ju nezrenovujeme, tak samozrejme ušetríme nejaké doláre.

Líši sa však táto úloha až tak veľmi od úlohy predchádzajúcej? Cesty predsa majú len dve možné ceny. Znie prirodzene, že ak chceme za rekonštrukciu zaplatiť čo najmenej peňazí, chceme zrekonštruovať čo najviac ciest s cenou 1. Najlepšie by dokonca bolo, ak by sme nemuseli zrekonštruovať žiadnu cestu s cenou 2.

Čo sa stane ak použijeme postup z podúlohy b), ale spustíme ho iba pre cesty s cenou 1? Medzi niektorými mestami sa bude dať prechádzať a získame tak niekoľko komponentov. V prípade, že nám ostane iba jeden komponent, tak algoritmus môžeme ukončiť, našli sme riešenie s cenou $n-1$, čo je najmenšia cena, ktorú vieme získať. Čo však v prípade, že tých komponentov ostane viacero?

Uvedomme si, že žiadna cesta s cenou 1 nám už nepomôže. To že sme ju nezrekonštruovali znamená, že vedie medzi dvoma mestami, medzi ktorými sa vieme pohybovať len po zrekonštruovaných cestách. Zároveň sme však použili najväčší počet ciest s cenou 1, aké sme použiť mohli. Skúsili sme totiž každú jednu z nich a nezrekonštruovali sme iba tie, ktoré nám nespojili žiadne dva komponenty. Ostáva nám teda už len pospájať zvyšné komponenty pomocou ciest s cenou 2.

A to vieme spraviť úplne rovnako. Pre každú cestu s cenou 2 sa spýtame, či nám jej zrekonštruovanie pomôže, teda či jej koncové mestá ležia v tom istom komponente. Ak neležia, takúto cestu zrekonštruujeme, čím zmenšíme počet komponentov. Samozrejme, cesty s cenou 1, ktoré sme sa rozhodli zrekonštruovať, pokladáme pri cestách s cenou 2 za zrekonštruované.

```
postupne pre každú cestu s cenou 1 sprav:
    ak sa mestá na koncoch tejto cesty nachádzajú v rovnakom komponente
    (dá sa medzi nimi dostať iba po zrekonštruovaných cestách), tak:
        s touto cestou nič nerob
    ak sa nenachádzajú v rovnakom komponente, tak:
        zrekonštruuj túto cestu

postupne pre každú cestu s cenou 2 sprav:
    ak sa mestá na koncoch tejto cesty nachádzajú v rovnakom komponente
    (dá sa medzi nimi dostať iba po zrekonštruovaných cestách), tak:
        s touto cestou nič nerob
    ak sa nenachádzajú v rovnakom komponente, tak:
        zrekonštruuj túto cestu
```

Podúloha d)

Vymyslieť správne riešenie tejto podúlohy, kde má každá cesta inú cenu, by malo byť pomerne jednoduché. Samozrejme, že chceme uprednostňovať lacnejšie cesty. A nechceme vybrať cestu, ktorá spája dve mestá, ktoré už sú v tom istom komponente. Môžeme preto použiť riešenie z podúlohy b), akurát nebudeme cesty spracovávať v ľubovoľnom poradí, ale od najlacnejšej po najdrahšiu. Aj toto je však veľmi ľahko zapísateľné.

```
usporiadaj všetky cesty od najlacnejšej po najdrahšiu
postupne pre každú cestu v usporiadanom poradí sprav:
    ak sa mestá na koncoch tejto cesty nachádzajú v rovnakom komponente
    (dá sa medzi nimi dostať iba po zrekonštruovaných cestách), tak:
        s touto cestou nič nerob
    ak sa nenachádzajú v rovnakom komponente, tak:
        zrekonštruuj túto cestu
```

Toto riešenie je dokonca také isté ako riešenie v podúlohe c), aj tam totiž s cestami robíme zakaždým to isté, a spracovávame ich v utriedenom poradí. Najskôr všetky s cenou 1, potom všetky s cenou 2.

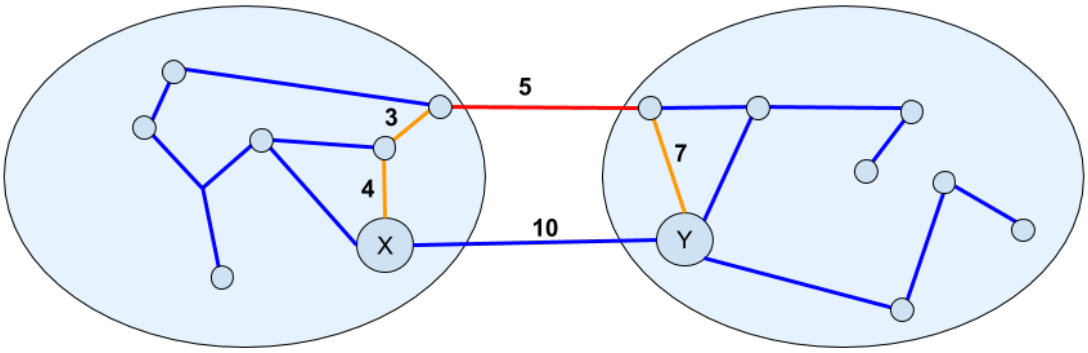
To, čo je ťažké na tejto podúlohe je dokázať, že takéto riešenie je správne a naozaj nájde najlacnejšie možné riešenie.

Predpokladajme, že **naše** riešenie je zlé. To znamená, že existuje iné, **optimálne** riešenie, v ktorom za zrekonštruovanie zaplatíme najmenšie množstvo dolárov. Vieme, že aj toto riešenie musí rekonštruovať $n - 1$ ciest, rovnako koľko zrekonštruuje aj naše riešenie[1]. Tieto dve riešenia sa však v niečom musia líšiť.

Predpokladajme, že optimálne riešenie zrekonštruuje cestu medzi mestami `X` a `Y`, ale naše riešenie túto cestu nezrekonštruuje. Prečo to nespraví? Jediný dôvod nezrekonštruovať túto cestu, má naš algoritmus v prípade, ak v okamihu keď sa rozhoduje o zrekonštruovaní tejto cesty, patria mestá `X` a `Y` do rovnakého komponentu. To ale znamená, že v Absurdistane existuje medzi mestami `X` a `Y` taká **trasa**, ktorá vedie iba po cestách lacnejších ako cesta z `X` do `Y`.

Odkiaľ sme toto zistili? Cesty predsa spracovávame v usporiadanom poradí od najlacnejších. Takže ak sa v našom riešení mestá `X` a `Y` nachádzali v tom istom komponente (a preto sme nezrekonštruovali cestu medzi `X` a `Y`), tak medzi týmito mestami sa dalo dostať iba po zrekonštruovaných, a teda lacnejších cestách.

Čo to znamená, pre naše optimálne riešenie? Odstráňme z neho cestu medzi `X` a `Y`. V takomto riešení sa teraz medzi týmito dvoma mestami nedá dostať. Musíme teda pridať nejakú cestu, ktorou to opravíme. Zoberme si iba cesty z **trasy** medzi `X` a `Y`, ktorá obsahuje iba cesty lacnejšie ako cesta medzi `X` a `Y`. Aspoň jedna z týchto ciest toto pokazené optimálne riešenie opraví. Spraví ale aj niečo viac. Keďže pridaná cesta bola lacnejšia ako cesta, ktorú sme odstránili, vytvorili sme riešenie, ktoré je ešte lepšie ako optimálne riešenie. Ale to predsa nie je možné! Čo to znamená? Že sme sa pomýlili v úplne prvom predpoklade, ktorý tvrdil, že naše riešenie je zlé. Naše riešenie je teda naozaj správne a nájde najlacnejšie možné riešenie.



Na obrázku je znázornené to, čo som sa snažil vysvetliť v predchádzajúcom odstavci. Modrou farbou sú znázornené cesty, ktoré zrekonštruuje optimálne riešenie. Oranžové (a červená) cesty označujú trasu medzi mestami `X` a `Y`, ktoré obsahujú iba cesty lacnejšie ako priama cesta medzi týmito dvoma mestami. Takúto cestu nám zaručuje naše riešenie a fakt, že naše riešenie nerekonštruuje cestu medzi `X` a `Y`. Všimnite si, že ak vymeníme modrú cestu s cenou 10 za červenú cestu s cenou 5, dostaneme riešenie, v ktorom sa stále dá dostať odvšadiaľ všade, ale naviac je o 5 miliónov dolárov lacnejšie.

Záver

Problém, ktorý ste riešili v tomto príklade je problém grafový a volá sa **hľadanie najlacnejšej kostry**. A

[1] Všimnite si, že tieto dve riešenia označujem ako naše a optimálne.

algoritmus, ktorý sme si tu predviedli sa skutočne používa, napríklad keď telekomunikácie potrebujú zistiť, kam zakopať káble tak, aby ich to vyšlo čo najlacnejšie, ale aj pri veľa iných problémoch. Názov tohto algoritmu je *Kruskalov algoritmus*.

V skutočnosti je ešte o chlp komplikovanejší. Aj keď nám ľuďom sa zdá podmienka **`ak sa dve mestá nachádzajú v rovnakom komponente`** vcelku jednoduchá a zrozumiteľná, počítačom to až tak jasné nie je. To čo ste vyriešili v tejto úlohe je preto iba polovica problému, druhá polovica sa zaujíma o to, ako má počítač zistiť, či sú dve mestá v tom istom komponente a ako to má zistiť dostatočne rýchlo. To však na riešenie tejto úlohy nebolo potrebné, možno si vás to nájde v nejakom ďalšom kole :)

2. Pokročilé prototypy

vzorák napísal Roman
(max. 15 b za riešenie)

Podúloha a) NP – stroj

Stroj sa veľmi podobá na stroj `P` zo zadania. Tento stroj ale vracia počet **jednotiek**, ktoré sú na vstupnej páske. Aby sme ho mohli použiť na vytvorenie stroja `NP`, potrebujeme na vstupnej páske vymeniť nuly za jednotky a naopak, jednotky za nuly. Na to slúži stroj `NOT`, ktorý poznáme z minulého kola.

Preto na vstupnú pásku použijeme stroj `NOT` a jeho výsledok vložíme do `P`.

```
Páska A:            0001110001101100
B = !A              1110001110010011
C = P(B)            0000000111111111 (=9)
Vytlač C:           0000000111111111 (=9)
```

Alebo v skratke `C = P(!A)`.

Podúloha b) NULP – stroj

Najskôr vytvorme o niečo jednoduchší stroj, ktorý zistí, koľko 1 má napravo od seba ďalšiu 1. Pretože nás zaujíma, či je pravý sused ľubovoľnej 1 opäť 1, skúsme celú pásku posunúť o 1 doľava pomocou stroja `LSHIFT`, resp. `<<`.

```
Páska A:            1100100111101011
B = A << 1          1001001111010110
```

Všimnime si, že páska `B` je teraz páskou pravých susedov `A` – na i-tej pozícií obsahuje pravého suseda i-tej cifry z `A`. Pozície, na ktorých je 1 na oboch páskach, sú preto hľadaným riešením. Sú to totiž pozície, na ktorých je 1 (páska `A`), a zároveň má napravo od seba 1 (páska `B`). Na pásky `A` a `B` preto použijeme stroj `AND`, ktorý ponechá 1 iba na miestach, kde ich obsahovali obe pásky. Výsledok následne spočítame pomocou stroja `P`.

```
Páska A:            1100100111100011
B = A << 1          1001001111000110
C = A & B           1000000111000010
D = P(C)            0000000000011111 (=5)
```

Teraz sa vráťme k pôvodnej úlohe. Tou bolo zistiť počet 0, ktoré majú napravo od seba 0. Zjavne stačí prehodiť 1 a 0 a môžeme použiť riešenie uvedené vyššie. Na to použijeme opäť stroj `NOT`.

```
Páska A:            1100100111100011
B = !A              0011011000011100
C = B << 1          0110110000111000
D = B & C           0010010000011000
E = P(D)            0000000000001111 (=4)
Vytlač E
```

Alebo v skratke `E = P(!A & (!A << 1))`

Poznámka: Namiesto posunu doľava `<<` sme mohli použiť aj posun doprava `>>`. Rozmyslite si, že na ľubovoľnej páske je rovnako veľa 0, ktoré majú napravo od seba 0, ako 0, ktoré majú naľavo od seba 0.

Podúloha c) POL – stroj

Najskôr zistíme počet 1 strojom `P`. Získané číslo potrebujeme vydeliť dvomi. Použijeme na to nasledovný trik: do stroja `AND` vložíme výstup `P(A)` a pásku 1010101010101010.

```
Páska A:                1001101011100100
B = P(A)                0000000011111111 (=8)
C = 1010101010101010    1010101010101010
D = B & C               0000000010101010
E = P(D)                0000000000001111 (=4)
```

Čo takýto stroj robí? Zo súvislého úseku jednotiek na páske `B=P(A)` vyberie iba tie na párnych pozíciách, teda polovicu z pôvodného počtu. Nakoniec ešte použijeme stroj `P`, aby sme tieto jednotky zarovnali doprava a vytvorili tak korektné číslo.

Iná predstava je, že pôvodné číslo rozdelíme na kôpky veľkosti 2, pretože ak páska `D` obsahuje na i-tej pozícií 1 znamená to, že páska `B=P(A)` obsahovala na i-tej a $i+1$ pozícií 1, ktoré spolu tvoria kôpku. Počet takýchto kôpok je potom pôvodné číslo vydelené dvomi.

Uvedomme si, že takéto riešenie navyše zaokrúhľuje nadol pri delení nepárnych čísel:

```
Páska A:                1000000011100100
B = P(A)                0000000000011111 (=5)
C = 1010101010101010    1010101010101010
D = B & C               0000000000001010
E = P(D)                0000000000000011 (=2)
```

Pri nepárnom čísle totiž najľavejšia jednotka na páske `B` vôbec neovplyvní výsledok. Je na nepárnej pozícií, pričom páska 1010101010101010[2] má na nepárnych pozíciách vždy 0. Ak by sme pri delení chceli zaokrúhľovať nahor, samozrejme by sme použili pásku `0101010101010101`.

V skratke: `E = P(P(A) & (1010101010101010))`

Podúloha d) TRI – stroj

Najskôr musíme vymyslieť spôsob, ako rozpoznať úseky dĺžky 3. Všimnime si, že najľavejšia 1 ľubovoľného úseku dĺžky aspoň 3 musí mať cifru 1 o jednu aj dve pozícii napravo od seba. Niečo podobné sme už riešili v prvej podúlohe. Aby sme sa dostali k cifre o jedna napravo, posunuli sme celú pásku o jednu pozícii doľava (`<< 1`). Ak teda chceme cifry o dve pozície napravo, posunieme pásku o dve cifry doľava (`<< 2`).

```
Páska A:            1110011110011100
B = A << 1          1100111100111000 (susedia napravo)
C = A << 2          1001111001110000 (susedia o 2 pozície napravo)
D = A & B           1100011100011000 (pozície s 1, ktoré majú 1 aj napravo od seba)
E = D & C           1000011000010000 (pozície s 1, ktoré majú dve 1 napravo od seba)
```

Ak má páska `E` na i-tej pozícii 1, znamená to, že na pôvodnej páske `A` sú na pozíciách i, $i+1$ a $i+2$ všade 1. V opačnom prípade by aspoň jedna z vymenovaných pozícií obsahovala 0.

Jednotky, ktoré ostali na páske `E` však nie sú hľadaným výsledkom. Nás totiž zaujímajú iba úseky jednotiek, ktoré majú dĺžku **presne** tri. Každý takýto úsek je síce na páske `E` reprezentovaný 1, ale aj úsek jednotiek dĺžky 4 nám na pásku `E` pridá nejaké jednotky, presnejšie rovno 2. Uvedomme si však, že úseky jednotiek dĺhších ako 3, nám na páske `E` vytvoria nejaký úsek jednotiek dlhší ako 1. To znamená, že na páske `E` chceme nájsť počet osamotených 1 – takých, ktoré majú napravo aj naľavo od seba 0.

Pri tom si pomôžeme strojom `RAO` z minulého kola. Ten vedel takéto osamotené z pásky odstraňovať. Máme teda pásku `E` na ktorej sú dobré aj zlé 1 a pásku `RAO(E)`, v ktorej sme dobré 1 odstránili. Koľko je teda dobrých jednotiek? No predsa `P(E) - P(RAO(E))` – ak spočítame počet jednotiek na páske `E` a odčítame od neho počet jednotiek z pásky `RAO(E)` tak výsledok je práve počet dobrých jednotiek. Jediný problém je, ako spraviť odčítanie. Na to nám ale veľmi šikovne poslúži stroj `XOR`. Pozrite sa na príklad nižšie a rozmyslite si, ako takéto odčítavanie bude fungovať.

```
Páska A:            1110011111011100
B = A << 1          1100111110111000 (susedia napravo)
C = A << 2          1001111101110000 (susedia o 2 pozície napravo)
D = A & B           1100011100011000 (pozície s 1, ktoré majú 1 aj napravo od seba)
E = D & C           1000011100010000 (pozície s 1, ktoré majú dve 1 napravo od seba)
F = RAO(E)          0000011100000000
```

[2] Takýmto páskam, v skutočnom počítači číslam, hovoríme *bitová maska* (angl. *bitmask*).

```
G = P(E)            0000000000011111 (=5, počet 1 vrátane osamotených)
H = P(F)            0000000000000111 (=3, počet 1 bez osamotených)
I = G ^ H           0000000000011000
J = P(I)            0000000000000011 (=2, počet osamotených 1)
```

Skrátene: `E = A & (A << 1) & (A << 2)` a potom `J = P( P(E) ^ P(RAO(E)) )`

Podúloha e) NAJDL – stroj

Vytvoríme si najskôr pomocný stroj `SEK`, ktorý z každého úseku jednotiek vymaže najľavejšiu jednotku. Stroj `SEK`:

```
Páska A:            0100111011010110
B = A >> 1          0010011101101011
C = A & B           0000011001000010
Vytlač C
```

Pomocou stroja `SEK` budeme postupne skracovať úseky jednotiek na vstupnej páske. Po prvom použití tohto stroja zmiznú všetky úseky dĺžky 1. Po dvoch použitiach už na páske neostane nič z úsekov dĺžky 2, atď.. To znamená, že ak sa po x použitiach stroja `SEK` na páske nachádza nejaká 1, tak na pôvodnej vstupne páske musel byť úsek jednotiek dĺžky aspoň $x + 1$.

Budeme teda opakovať nasledovný postup. Najskôr zistíme, či sa na páske nachádza nejaká 1. Toto vieme ľahko spraviť pomocou stroja `P`. Na páske `B` si potom budeme pamätať, koľkokrát sa tam naozaj nejaká jednotka nachádzala. No a následne použijeme na pásku stroj `SEK`.

Na pásku `B` zapíšeme toľko jednotiek, aká bolo dĺžka najdlhšieho úseku jednotiek na vstupnej páske a vďaka tomu ľahko zistíme odpoveď.

```
Páska A:            0111101101111110
B = 0               0000000000000000
C = 1               0000000000000001

D = P(A)            0000111111111111 (=12)
E = D & C           0000000000000001 (A obsahuje aspoň 1 jednotku)
B = B OR E          0000000000000001 (pripíšem výsledok na pásku B)
A = SEK(A)          0011100100111110

D = P(A)            0000000111111111 (=9)
E = D & C           0000000000000001 (A obsahuje aspoň 1 jednotku)
B = B << 1          0000000000000010 (spravím si miesto pre novú informáciu)
B = B OR E          0000000000000011 (pripíšem výsledok na pásku B)
A = SEK(A)          0001100000011110

....
14x
....

B = P(B)
Vytlač B
```

Všimnite si, že sme použili výrazy typu `A = SEK(A)`. To znamená, že stroj `SEK` príjme pôvodnú hodnotu pásky `A` a svoj výsledok zapíše opäť na pásku `A`, čím zmení jej obsah.

Naše riešenie však používa tie isté operácie stále dookola. Pre jednoduchší zápis, môžeme použiť cyklus tvaru `opakuj k`, do ktorého vložíme nejaké operácie. Tieto operácie sa potom zopakujú k krát.

```
Páska A
B = 0
C = 1

Opakuj 16:
    D = P(A)
```

```
    E = D & C
    B = B << 1
    B = B OR E
    A = SEK(A)

B = P(B)
Vytlač B
```

Alebo ešte stručnejšie:

```
B = 0
Opakuj 16:
    B = B << 1
    B = B OR ( P(A) & 1 )
    A = SEK(A)

Vytlač P(B)
```

3. Polročné vysvedčenie

vzorák napísal Adam
(max. 15 b za riešenie)

Podúloha a)

Na škole sa vyučuje jeden neštandardný predmet. Napíšte nám, aké je jeho ID.

Z databázy si necháme vypísať zoznam všetkých predmetov, ktoré sa na škole vyučujú, pomocou príkazu

```
SELECT * FROM predmety;
```

V zozname predmetov potom nájdeme vskutku neštandardný predmet s názvom `Obrana proti čiernej mágii`. Vidíme, že jeho ID je 14.

Podúloha b)

Dostal niekto 24. 12. jednotku?

Keďže nás zaujímajú známky, má zmysel sa pozerať do tabuľky `znamky`. Ako vidíme, v tejto tabuľke je ku každej známke zaznačený aj dátum, v ktorom bola udelená a takisto jej hodnota. Nás zaujímajú iba jednotky, ktoré boli udelené 24.12., preto použijeme príkaz `WHERE`, pomocou ktorého vieme vybrať riadky s konkrétnymi hodnotami. Jednotlivé podmienky pospájame pomocou `AND`, keďže chceme aby platili všetky naraz.

```
SELECT *
FROM znamky
WHERE hodnotenie_den = 24 AND hodnotenie_mesiac = 12 AND hodnotenie = 1;
```

Hľadaný deň, mesiac a hodnotenie nepíšeme v úvodzovkách, pretože tieto polia majú formát čísla. Z odpovede v databáze vidíme, že v daný deň bolo udelených až 58 jednotiek.

Podúloha c)

Koľko žiakov sa volá Jozef alebo Juraj alebo má priezvisko Novák?

Hľadáme žiakov, takže budeme používať tabuľku `ziaci`. Vypísať však chceme len tých, ktorí majú meno Jozef **alebo** Juraj **alebo** majú priezvisko Novák. Na toto opäť použijeme príkaz `WHERE`, tentokrát ale jednotlivé časti pospájama slovom `OR`, pretože stačí, aby platila len jedna z podmienok.

```
SELECT *
FROM ziaci
WHERE meno = "Jozef" OR meno = "Juraj" OR priezvisko = "Novák";
```

Môžete si všimnúť, že vo výsledku je viacero Jozefov Novákov. Každý z nich má však unikátne ID, takže sú to odlišní žiaci s rovnakým meomm. Každý jednotlivý žiak s menom Jozef Novák je však vypísaný iba raz, napriek tomu, že spĺňa obe podmienky. To je spôsobené tým, ako databáza prehľadáva tabuľku – postupne prechádza všetky záznamy (žiakov) a pre každý záznam overuje zadanú podmienku. Ak podmienka platí, záznam vypíše.

Všimnite si, že jednotlivé mená sme museli dať do úvodzoviek. Odpoveďou je počet riadkov výsledku, teda 280.

Podúloha d)

Aký je vekový rozdiel medzi najmladším a najstarším žiakom?

V tejto podúlohe nás zaujímajú všetci žiaci, preto nebudeme musieť použiť príkaz `WHERE`. To čo potrebujeme je usporiadať žiakov podľa dátumu narodenia. Na to slúži príkaz `ORDER BY`. Pomocou neho vieme určiť, na základe ktorých vlastností sa má výsledok (v našom prípade celá tabuľka `ziaci`) zoradiť. Na nájdenie najstaršieho žiaka poslúži nasledovný príkaz.

```
SELECT *
FROM ziaci
ORDER BY narodeniny_rok ASC, narodeniny_mesiac ASC, narodeniny_den ASC;
```

Slovo `ASC` (teda stúpajúco) sme použiť nemuseli, keďže takéto zoradenie sa používa štandardne. Tak či tak, prvý riadok výsledku predstavuje najstaršieho žiaka, ktorý sa narodil 2.9.2000.

Na nájdenie najmladšieho žiaka sa potom buď pozrieme na spodok tabuľky, alebo príkaz otočíme pomocou `DESC` (klesajúco).

```
SELECT *
FROM ziaci
ORDER BY narodeniny_rok DESC, narodeniny_mesiac DESC, narodeniny_den DESC;
```

Zistíme, že najmladší žiak sa narodil 30.8.2009. Vekový rozdiel najmladšieho a najstaršieho žiaka je teda takmer 9 rokov.

Podúloha e)

Ktorý učiteľ učí 3.B matematiku?

Hľadáme meno učiteľa, ktorý učí 3.B. matematiku. Na to by nám mohla poslúžiť tabuľka `vyucovania`, v ktorej sú záznamy o tom, ktorý učiteľ učí ktorú triedu ktorý predmet. Jediný problém je, že v tejto tabuľke sa nachádajú iba IDčka učiteľov, tried a predmetov.

Aby sme k ID vedeli priradiť aj konkrétne meno, musíme použiť aj tabuľky `triedy`, `ucitelia` a `predmety`. Tieto tabuľky potrebujeme spojiť na základe rovnakých ID, čo spravíme pomocou príkazu `WHERE`. Napríklad bude musieť platiť, že `vyucovania.id_triedy = triedy.id`.

Ako sa už písalo v tutoriáli, keď do `FROM` zadáme viacero tabuliek, ich obsahy sa skombinujú. Každý riadok jednej tabuľky sa spojí s každým riadkom druhej. V našom prípade dokonca dostaneme až štvorice riadkov. Veľa z nich nám však vôbec nepomôže. Získať záznam, ktorý spája učiteľa Jánoša (ID 3) s vyučovaním, ktoré učil Boška (ID 8), je asi zbytočné. Keďže sa však ich IDčka líšia, vieme ich ľahko vyfiltrovať vo `WHERE`. Preto nám ostanú iba zmysluplné záznamy, ktoré spájajú učiteľov s ich vyučovaniami. A samozrejme rovnako to funguje aj pre `ucitelia` a `predmety`.

Naviac vo `WHERE` upresníme aj to, že hľadáme iba triedu 3.B. a predmet matematika.

```
SELECT predmety.meno, triedy.meno, ucitelia.meno, ucitelia.priezvisko
FROM vyucovania, ucitelia, triedy, predmety
WHERE ucitelia.id = vyucovania.id_ucitela AND vyucovania.id_predmetu = predmety.id
AND vyucovania.id_triedy = triedy.id AND triedy.meno = "3.B" AND predmety.meno = "Matematika";
```

Vidíme, že v tomto prípade sme za `FROM` potrebovali dosadiť až štyri tabuľky. Takisto aby výsledok nebol príliš komplikovaný, `SELECT` nám určil, ktoré stĺpce nás zaujímajú. V tomto prípade iba meno triedy, predmetu a meno učiteľa (stačilo by síce vypísať iba meno učiteľa, takto si to však vieme skontrolovať).

Výsledkom je učiteľ Alfréd Gašparovič.

Podúloha f)

Napíš mená všetkých učiteľov matematiky.

Postup je rovnaký ako v predošlej podúlohe, akurát nepotrebujeme použiť tabuľku `triedy`. To, ktorú triedu daný učiteľ učí nás totiž nezaujíma, dôležité je len to, že učí matematiku.

```
SELECT ucitelia.meno, ucitelia.priezvisko
FROM vyucovania, ucitelia, predmety
WHERE ucitelia.id = vyucovania.id_ucitela AND vyucovania.id_predmetu = predmety.id
AND predmety.meno = "Matematika";
```

Výsledkom je tabuľka 33 učiteľov. Je tam však drobný problém. Ak sa lepšie pozrieme na mená týchto učiteľov, zistíme, že niektoré z nich sa opakujú. Keď sa však nad tým zamyslíme, nie je to až tak prekvapivé. Riadky totiž vyberáme na základe rôznych riadkov `vyucovania`. Ale jeden učiteľ matematiky učí matematiku pravdepodobne viacero tried. Za každú triedu (teda iný riadok v tabuľke `vyucovania`), ktorú nejaký učiteľ učí matematiku sa tento učiteľ dostane do výsledku.

Aby sme teda zistili, koľko rôznych učiteľov učí na našej škole matematiku, musíme z výsledných záznamov odstrániť duplikáty. To môžeme spraviť ručne, pričom nám veľmi pomôže, ak si do nášho príkazu pridáme `ORDER BY ucitelia.priezvisko`. Alebo na to môžeme použiť už existujúci príkaz SQL – `SELECT DISTINCT`. Ten funguje úplne rovnako ako príkaz `SELECT`, ale v získanom výsledku navyše odstráni opakujúce sa riadky.

```
SELECT DISTINCT ucitelia.meno, ucitelia.priezvisko
FROM vyucovania, ucitelia, predmety
WHERE ucitelia.id = vyucovania.id_ucitela AND vyucovania.id_predmetu = predmety.id
AND predmety.meno = "Matematika";
```

Zistíme, že na škole učí matematiku 13 učiteľov.

Podúloha g)

Ktorý študent dostal naposledy päťku?

Tabuľka `znamky` obsahuje iba ID žiakov. Potrebujeme si tieto spojiť tabuľky `znamky` a `ziaci` tak, ako v predchádzajúcich úlohách cez rovnosť IDčok. Takisto do príkazu `WHERE` zadáme, že hľadáme iba päťky a výsledok zoradíme príkazom `ORDER BY`, aby sa nám najnovšia päťka zobrazila ako prvý riadok výsledku.

```
SELECT ziaci.meno, ziaci.priezvisko, znamky.hodnotenie,
hodnotenie_rok, hodnotenie_mesiac, hodnotenie_den
FROM znamky, ziaci
WHERE znamky.id_ziaka = ziaci.id AND hodnotenie = 5
ORDER BY hodnotenie_rok DESC, hodnotenie_mesiac DESC, hodnotenie_den DESC;
```

Ako vidíme, pätiek bolo udelených posledný deň v škole pomerne dosť. Najjednoduchší spôsob ako ponechať vo výsledku iba tieto pätky, je do `WHERE` pridať podmienku na daný rok, mesiac a deň. Naviac si musíme dať pozor, aby sme niekoho nezarátali viackrát, na čo nám opäť poslúži `SELECT DISTINCT`. Ani to však nestačí, pretože ak sú nejaký dvaja žiaci menovci, tak ich `SELECT DISTINCT` prehlási za duplikáty. Aby sme sa tomu vyhli, pridáme do výsledku aj `ziaci.id`. Dvaja ľudia s tým istým menom budú mať totiž rôzne ID.

```
SELECT DISTINCT ziaci.id, ziaci.meno, ziaci.priezvisko, znamky.hodnotenie,
hodnotenie_rok, hodnotenie_mesiac, hodnotenie_den
FROM znamky, ziaci
WHERE znamky.id_ziaka = ziaci.id AND hodnotenie_rok = 2016 AND hodnotenie_mesiac = 6
AND hodnotenie_den = 30 AND hodnotenie = 5;
```

Zistíme, že týchto pätiek bolo rozdaných 73 rôznym študentom.

Podúloha h)

Ktorý študent dostal naposledy päťku z matematiky od profesorky Vlnovej?

Táto podúloha sa síce veľmi ponáša na tú predchádzajúcu, predsa je však o dosť náročnejšia.

V predchádzajúcej podúlohe sme spojili tabuľky `znamky` a `ziaci`, vďaka čomu sme o každej známke zistili aj to, ktorý konkrétny žiak ju získal. Naviac sme si nechali iba päťky. Teraz však potrebujeme zistiť aj to, ktorý učiteľ túto známku dal. Bohužiaľ, v tabuľke `znamky` nie je `id_ucitela`, jediné, čím si vieme pomôcť je `id_predmetu`. Pripojme teda tabuľku `predmety`, opäť podľa rovnosti IDčok. Vďaka tomu vieme do `WHERE` pridať podmienku `predmety.meno = „Matematika"`, takže nám ostanú už iba pätorky z matematiky.

Matematiku však môže na škole učiť aj viacero učiteľov. Ako teda zistíme, ktoré pätky dala Vlnová? Určite však platí, že konkrétnu triedu učí matematiku iba jeden učiteľ. A tabuľka `vyucovania` nám podáva presne túto

informáciu. Preto do riešenie pripojíme tabuľku `vyucovania`, pričom si musíme dať pozor, aby sme porovnávali aj `znamky.id_predmetu` a aj `ziaci.id_triedy`.

V tomto momente už o každej známke vieme z akého predmetu bola udelená, v ktorej triede, ktorému žiakovi a takisto poznáme ID učiteľa, ktorý ju dal. Stačí teda podľa `vyucovania.id_ucitela` pripojiť tabuľku `ucitelia` a vybrať iba známky, ktoré udelila učiteľka s priezviskom Vlnová.

```
SELECT *
FROM znamky, ziaci, predmety, vyucovania, ucitelia
WHERE znamky.id_ziaka = ziaci.id AND znamky.hodnotenie = 5
AND znamky.id_predmetu = predmety.id AND predmety.meno = "Matematika"
AND znamky.id_predmetu = vyucovania.id_predmetu
AND ziaci.id_triedy = vyucovania.id_triedy AND vyucovania.id_ucitela = ucitelia.id
AND ucitelia.priezvisko = "Vlnová"
ORDER BY znamky.hodnotenie_rok DESC, znamky.hodnotenie_mesiac DESC,
znamky.hodnotenie_den DESC;
```

Samozrejme napísať takýto zložitý výraz je náročné. Aj pri písaní sa však dalo postupovať po krokoch, presne ako vo vzorovom riešení. Stačilo tabuľky pridávať jednu po druhej, vždy si overiť, že aktuálny príkaz dáva správny výsledok a až potom pridať ďalšiu tabuľku a ďalšiu podmienku. Dôležité len bolo nezľaknúť sa.

Zistíme, že naposledy dostali päťku z matematiky od učiteľky Vlnovej žiaci: Edita Melichárová, Kamil Lipa a Erik Loja.

Podúloha i)

V ktorom predmete sa žiakom najviac darilo v septembri? Teda taký, z ktorého bola najlepšia priemerná známka za september?

V tejto podúlohe potrebujeme zistiť, v ktorom predmete mali žiaci v septembri najlepšiu **priemernú** známu. Čo znamená, že budeme musieť použiť príkaz `GROUP BY`.

Ak si zoberieme všetky známky, je ľahké z nich vybrať tie, ktoré boli udelené v septembri. Následne ich chceme rozdeliť do skupín podľa predmetu. A práve na to slúži príkaz `GROUP BY predmety.id`. Názvy stĺpcov za `GROUP BY` udávajú, podľa ktorých hodnôt sa majú tieto riadky zoskupiť. Následne vieme na zvyšné stĺpce použiť agregačnú funkciu, akou je napríklad primer `AVG`. Dostávame pomerne jednoduchý príkaz.

```
SELECT predmety.meno, AVG(znamky.hodnotenie) AS priemer
FROM znamky, predmety
WHERE znamky.id_predmetu = predmety.id AND znamky.hodnotenie_mesiac = 9
GROUP BY predmety.id
ORDER BY priemer;
```

Všimnite si, že na konci sme výsledok usporiadali podľa vypočítaného priemeru. Aby sme mohli za `ORDER BY` dať tento priemer, museli sme si príslušný stĺpec pomenovať týmto menom, na čo slúžil príkaz `AS`.

Odpoveďou je Prvouka s priemerom 2.8934.

Podúloha j)

Ktorá trieda má najlepší priemer známok z matematiky?

Po všetkých predchádzajúcich úlohách by nám táto už nemala robiť žiadne problémy. Spojením tabuliek `znamky` a `ziaci` sa dozvieme, ktorú triedu navštevuje žiak, ktorý dostal konkrétnu známku. Aby sme naviac nemuseli pracovať iba s IDčkami tried a predmetov, pridáme si k nim tabuľky `predmety` a `triedy`.

Teraz už vieme veľmi jednoducho ponechať iba známky z matematiky a pomocou `GROUP BY` ich rozdeliť do skupín podľa jednotlivých tried (je jedno či na to použijeme `triedy.id`, `triedy.meno` alebo `ziaci.id_triedy`). Následne vieme pre každú triedu získať priemernú známku pomocou `AVG`.

```
SELECT triedy.meno, AVG(znamky.hodnotenie) AS priemer
FROM znamky, ziaci, predmety, triedy
WHERE znamky.id_ziaka = ziaci.id AND id_predmetu = predmety.id
AND predmety.meno = "Matematika" AND ziaci.id_triedy = triedy.id
GROUP BY ziaci.id_triedy
ORDER BY priemer;
```

S priemerom 2.7733 vyhrá trieda 9.B..

Podúloha k)

Ktorý učiteľ nadŕža dievčatám? Teda taký, ktorý dáva viac jednotiek dievčatám ako chlapcom? Zaujíma nás učiteľ, ktorý dal napríklad dievčatám celkovo 150 jednotiek a chlapcom iba 93.

A prichádza na rad posledná (nerátajúc bonus) podúloha. A jej riešenie dalo skutočne zabrať. Pri jej riešení ste totiž museli buď mierne improvizovať, alebo si naštudovať o SQL ešte niečo naviac. Poďme si teda ukázať, ako sa to dalo vyriešiť.

Ako prvé musíme vedieť spočítať, koľko jednotiek rozdal daný učiteľ chlapcom a koľko dievčatám. Zamerajme sa najskôr na chlapcov[3]. Aj pomocou predchádzajúcich úloh by sme mali vedieť vytvoriť tabuľku, v ktorej sa budú nachádzať všetky jednotky, ktoré dostali chlapci a pri nich aj meno učiteľa, ktorý túto známku dal.

Rovnako ako v podúlohe h) musíme správne spojiť tabuľky `znamky`, `ziaci`, `vyucovania` a `ucitelia`. Pozor si musíte dať hlavne na to, aby ste ošetrili rovnosť všetkých troch hodnôt z tabuľky `vyucovania`, teda `id_ucitela`, `id_triedy` a `id_predmetu`. Inak sa vám môže stať, že zarátate známku, ktorú dal daný učiteľ z rovnakého predmetu v inej triede. Alebo v prípade, že učiteľ učí v tej istej dva predmety a vy neošetríte `vyucovania.id_predmetu = znamky.id_predmetu`, budete jednotky z nich rátať dvakrát. Následne pomocou `WHERE` ľahko ponecháme iba jednotky, ktoré dostali chlapci.

Teraz chceme tieto jednotky zoskupiť podľa učiteľa, ktorý ich dal. Do `GROUP BY` preto musíme dať mená stĺpcov, podľa ktorých chceme riadky zoskupovať. Najlepšie bude použiť samotné IDčka učiteľov, aby sme sa vyhli problémom, keď je na škole viacero učiteľov s tým istým menom aj priezviskom. No a ako bolo napísané v tutoriáli, na spočítanie riadkov výsledku slúži agregačná funkcia `COUNT`. Keďže sa však jedná o jednotky, rovnako dobre by sme mohli použiť `SUM`, keďže súčet jednotiek je ich počet. Dostávame nie až tak komplikovaný príkaz.

```
SELECT ucitelia.meno, ucitelia.priezvisko, COUNT(znamky.hodnotenie) AS pocet_chlapci
FROM znamky, ziaci, vyucovania, ucitelia
WHERE znamky.id_ziaka = ziaci.id AND ziaci.id_triedy = vyucovania.id_triedy
AND vyucovania.id_ucitela = ucitelia.id AND ziaci.pohlavie = "C"
AND znamky.hodnotenie = 1 AND vyucovania.id_predmetu = znamky.id_predmetu
GROUP BY ucitelia.id;
```

Vytvoriť rovnakú tabuľku aj pre dievčatá následne nie je problém. Stačí ak `"C"` zmeníme za `"D"`. Ako však porovnáme počty jednotiek, ktoré dostali chlapci a dievčatá?

Ak ste sa rozhodli improvizovať, pomôcť vám môže tabuľkový kalkulátor, akým je napríklad Excel. Jednoducho si necháte vypísať obe tabuľky, skopírujete ich do tabuľkového kalkulátora a v ňom už veľmi jednoducho viete spočítať rozdiel dvoch čísel vo všetkých riadkoch. Len si musíte dať pozor, aby ste dali k sebe výsledky pre tých istých učiteľov.

Takto ste však ručne urobili niečo, čo celý čas za vás robilo SQL. Predsa sa to musí dať spraviť aj v ňom. A veruže dá. Odpoveďou sú **dočasné tabuľky**. Uvedommte si, že vždy keď ste spustili nejaký príkaz `SELECT`, tak výsledkom bola tabuľka. Niekedy prázdna, niekedy len s jedným riadkom a občas ich obsahovalo vyše 20 000. Ale zakaždým to bola iba nejakým spôsobom zmodifikovaná tabuľka.

Pre programátora by teda malo byť prirodzené, že si vie takúto tabuľku odložiť na neskoršie použitie, ako to robí bežne s premennými. Či už natrvalo, alebo iba dočasne, kým mu neskončí aktuálny výpočet. To mu okrem iného aj umožní rozdeliť svoju prácu. Namiesto toho aby písal jeden obrovský SQL príkaz, môže napísať niekoľko kratších, zrozumiteľnejších a prehľadnejších, ktoré nadväzujú jeden na druhý. A ako vidíme v tejto úlohe, niekedy to bez použitia pomocných tabuliek ani nejde.

No a na toto slúži príkaz `CREATE VIEW meno AS`. Ten vytvorí dočasnú tabuľku, do ktorej si môžete uložiť svoje medzivýpočty. Jeho použitie je veľmi ľahké. Ak chcete vytvoriť tabuľku, použijete tento príkaz, doplníte meno, ktorým chcete túto tabuľku nazývať a za `AS` dáte príkaz `SELECT`, ktorý hovorí, ako túto tabuľku naplniť.

Vďaka tomu si vieme vytvoriť dve tabuľky, jednu pre počty jednotiek chlapcov (`chlapci`), druhú pre počty jednotiek dievčat (`dievcata`). Potom tieto tabuľky spojíme podľa mien učiteľov a do `SELECT` dáme vypísať rozdiel počtu jednotiek v tabuľke `dievcata` a v tabuľke `chlapci`. Zistíme, že najviac nadržiava dievčatám Dávid Kočka, ktorý im dal až o 38 viac jednotiek ako chlapcom.

```
CREATE VIEW chlapci AS
SELECT ucitelia.meno, ucitelia.priezvisko, COUNT(znamky.hodnotenie) AS pocet_chlapci
FROM znamky, ziaci, vyucovania, ucitelia
```

[3] Nech nás nemôžu obviniť, že nadržiavame

```
WHERE znamky.id_ziaka = ziaci.id AND ziaci.id_triedy = vyucovania.id_triedy
AND vyucovania.id_ucitela = ucitelia.id AND ziaci.pohlavie = "C"
AND znamky.hodnotenie = 1 AND vyucovania.id_predmetu = znamky.id_predmetu
GROUP BY ucitelia.id;

CREATE VIEW dievcata AS
SELECT ucitelia.meno, ucitelia.priezvisko, COUNT(znamky.hodnotenie) AS pocet_dievcata
FROM znamky, ziaci, vyucovania, ucitelia
WHERE znamky.id_ziaka = ziaci.id AND ziaci.id_triedy = vyucovania.id_triedy
AND vyucovania.id_ucitela = ucitelia.id AND ziaci.pohlavie = "D"
AND znamky.hodnotenie = 1 AND vyucovania.id_predmetu = znamky.id_predmetu
GROUP BY ucitelia.id;

SELECT chlapci.meno, chlapci.priezvisko,
dievcata.pocet_dievcata - chlapci.pocet_chlapci AS rozdiel
FROM chlapci, dievcata
WHERE chlapci.meno = dievcata.meno AND chlapci.priezvisko = dievcata.priezvisko
ORDER BY rozdiel DESC;
```

Poznámka k dočasným tabuľkám a nášmu webovému rozhraniu: Ak v našom webovom rozhraní spustíte príkaz `CREATE VIEW`, vytvorí sa dočasná tabuľka, ktorá tam bude až do momentu kým nerefreshnete stránku. To znamená, že ak sa pokúsite druhýkrát spustiť príkaz `CREATE VIEW` s tým istým menom bez toho, aby ste stránku refreshli, program vám vypíše chybu, lebo taká tabuľka už bude existovať a on ju nemôže len tak prepísať.

Bonusová úloha

Ktorý učiteľ dáva priemerne lepšie známky triede, ktorej je aj triedny učiteľ? Teda je to učiteľ, ktorý nadŕža svojim študentom.

No čo? Zistili ste ktorý učiteľ najviac nadržiaval svojim?

Tak ako v predchádzajúcej úlohe, aj tu si budeme musieť pomôcť dočasnými tabuľkami. V jednej pre každého učiteľa zistíme, aké dáva priemerné známky žiakom svojej triedy a v druhej, aké dáva priemerné známky žiakom mimo svojej triedy.

Toto nie je až také ťažké a v mnohom sa to podobá na to, čo sme už robili v predchádzajúcich úlohách. Naviac však budeme používať hodnotu `triedy.id_ucitela`, v ktorej je uložené ID triedneho učiteľa každej triedy. Takisto za zmienku stojí, že prvýkrát použijeme negáciu, keď budeme chcieť, aby sa trieda, do ktorej žiak chodí **ne**rovnala triede, ktorej je daný učiteľ triedny. Na to poslúži príkaz `NOT`.

Takto získané dve tabuľky potom spojíme na základe mien učiteľov a príkazom `WHERE` vyberieme všetky riadky, kde je priemer známok triedy učiteľa nižší ako priemer zvyšných žiakov, ktorých učí. Zistíme, že nadŕža až 24 učiteľov, pričom najväčší vinník je Urban Borka (2.8354 oproti 3.0784) a najmenší vinník je Levoslav Franko (3.0063 oproti 3.0083).

```
CREATE VIEW triedny_ucitel AS
SELECT ucitelia.id, ucitelia.meno, ucitelia.priezvisko,
AVG(znamky.hodnotenie) AS priemerna_znamka_trieda
FROM znamky, ucitelia, vyucovania, ziaci, triedy
WHERE vyucovania.id_ucitela = ucitelia.id AND vyucovania.id_triedy = ziaci.id_triedy
AND ziaci.id_triedy = triedy.id AND triedy.id_ucitela = ucitelia.id
AND znamky.id_ziaka = ziaci.id AND znamky.id_predmetu = vyucovania.id_predmetu
GROUP BY ucitelia.id;

CREATE VIEW iny_ucitel AS
SELECT ucitelia.id, ucitelia.meno, ucitelia.priezvisko,
AVG(znamky.hodnotenie) AS priemerna_znamka_netrieda
FROM znamky, ucitelia, vyucovania, ziaci, triedy
WHERE vyucovania.id_ucitela = ucitelia.id AND vyucovania.id_triedy = ziaci.id_triedy
AND ziaci.id_triedy = triedy.id AND NOT triedy.id_ucitela = ucitelia.id
AND znamky.id_ziaka = ziaci.id AND znamky.id_predmetu = vyucovania.id_predmetu
GROUP BY ucitelia.id;
```

```
SELECT *
FROM triedny_ucitel, iny_ucitel
WHERE triedny_ucitel.id = iny_ucitel.id
AND triedny_ucitel.priemerna_znamka_trieda < iny_ucitel.priemerna_znamka_netrieda
ORDER BY iny_ucitel.priemerna_znamka_netrieda - triedny_ucitel.priemerna_znamka_trieda DESC;
```

Záver

Dúfame, že sa vám úloha s SQL páčila. Je to rozhodne zaujímavý a užitočný programovací jazyk. Webový portál s databázou ostane naďalej prístupný, takže si kľudne môžete vyskúšať programy zo vzorového riešenia, poprípade zistiť nejakú ďalšiu zaujímavú vlastnosť, ktorú učitelia na tejto škole majú.

No a koho to zaujíma, tak dáta v tejto databáze boli samozrejme umelé. Aj mená vznikli iba náhodnou kombináciou mien a priezvisk z voľne dostupných slovníkov. Ako však vidíte, aj z takejto malej a pomerne jednoduchej databázy sa dali získať vcelku zaujímavé dáta.