
MO–P 2003

Soutěž dětí a mládeže v programování

kategorie starší žáci

obvodní kolo — obvod Praha 1

Zadání a řešení úloh

Tento text se pokouší vyložit řešení úloh řešených v obvodním kole soutěže v programování kategorie starší žáci, pořádané v rámci Prahy 1. Nejde o vyčerpávající vysvětlení a od čtenářů předpokládá určitou znalost programátorských postupů (alespoň takovou, jaká je očekávatelná od účastníků soutěže v programování).
Oficiální stránka oblastních kol soutěží v programování na Praze 1 je na

<http://voda.webz.cz/mo-p/index.html>

Organizace soutěže by se neobešla bez obětavé pomoci dalších lidí a subjektů, které si velice vážíme. Zejména bychom rádi poděkovali

Mgr. Dagmar Sejkorové, ředitelce Základní školy Uhelný trh,
v jejíchž prostorách soutěž zdárně proběhla,
Dr. Alexandře Svátové za pomoc při organizaci soutěže,
Ing. Jakubu Fischerovi za koordinaci soutěže.

Na organizaci soutěže a vydání tohoto sborníku přispěl **Magistrát hl. m. Prahy**.

1 Rychlé SMS

1.1 Zadání

Psaní SMS na mobilním telefonu bez pomůcek typu T9, apod. je poměrně zdlouhavé. Každá klávesa má na sobě v určitém pořadí písmena. Pokud chcete některá vložit, je potřeba klávesu stisknout opakovaně tak, aby počet stisků odpovídal pozici písmene v seznamu písmen (například klávesa 2 má na sobě napsáno „abc“; abychom vložili písmeno „c“, je ji potřeba stisknout třikrát, pro vložení písmene „a“ stačí jediný stisk).

Hodilo by se, kdyby byla písmena rozmístěna tak, aby napsání nějakého obvyklého textu znamenalo pokud možno co nejméně stisků. To nejde udělat univerzálně, protože každý světový jazyk obsahuje jiná písmena, ta se navíc v každém jazyce vyskytují s jinou frekvencí. Každé písmeno je přitom uvedeno na právě jedné klávese a písmena jsou na telefonu seřazena abecedně. Na první klávese tedy může být „a“, „ab“, „abc“ atd., nikoli však „abd“ apod., písmena na druhé klávese navazují v abecedě na písmena na klávese první. Počet písmen, která mohou být na jedné klávese, není nijak omezen. Napište program, který tuto úlohu zvládne pro jazyk s 10 písmeny a klávesnici se 4 tlačítky. Ze vstupu přečte pro každé písmeno průměrný počet výskytů ve sto znacích daného jazyka (bude to celé číslo větší než nula, součet všech četností je 100) a který určí a vypíše, jak mají být písmena rozdělena na 4 klávesy. Dále vypíše počet stisků, potřebných pro napsání 100 písmen průměrného textu.

Př.:

Vstup:

24
24
4
4
4
4
4
4
4
24

První a druhý znak se ve 100 znacích textu vyskytují průměrně 24krát, třetí, čtvrtý a pátý, šestý, sedmý, osmý a devátý se vyskytují průměrně 4krát. Desátý znak se vyskytuje průměrně 24krát.

Výstup:

Nejlepší rozložení znaků:

1. klávesa: 1-1
2. klávesa: 2-5
3. klávesa: 6-9
4. klávesa: 10-10

K napsání 100 znaků je potřeba v průměru 148 stisků.

Dvojice čísel oddělených pomlčkou u jednotlivých kláves označují, které znaky jsou na dané klávese umístěny. Na první klávese je pouze znak č. 1, na druhé klávese znaky č. 2, 3, 4 a 5, na třetí klávese znaky č. 6, 7, 8 a 9 a na poslední klávese je znak č. 10.

Jak jsme dospěli k číslu 148? První znak se ve 100 znacích běžného textu vyskytuje 24krát a je jako první na klávese číslo 1. Ke každému jeho napsání je nutné tuto klávesu jednou stisknout, pro napsání 24 výskytů tohoto znaku je potřeba ji stisknout 24krát. Podobně pro druhý znak. Třetí znak se vyskytuje ve 100 znacích 4krát a je na druhé pozici na druhé klávese. Ke každému vložení je potřeba dvou stisků druhé klávesy, pro vložení 4 výskytů je potřeba klávesu stisknout 8krát. Podobně pro čtvrtý znak, který je ale na třetí pozici na druhé klávese, takže během psaní 100 znaků běžného textu je nutné ji stisknout 12krát. Pokud stejným způsobem určíme i počty stisků pro další znaky, dostaneme

$$1 * 24 + 1 * 24 + 2 * 4 + 3 * 4 + 4 * 4 + 1 * 4 + 2 * 4 + 3 * 4 + 4 * 4 + 1 * 24 = 148$$

1.2 Řešení

U této úlohy není úplně snadné vymyslet třeba i hloupé řešení, zkusme popsat, co hledáme. Naším cílem je najít rozložení znaků na klávesy tak, aby počet stisků, potřebných pro napsání 100 znaků běžného textu byl minimální.

Především, jak zjistíme počet stisků pro napsání 100 znaků běžného textu? Tak, že sečteme počty stisků kláves, potřebných k napsání jednotlivých znaků, které se v těchto 100 znacích budou vyskytovat. Znaky, které se ve 100 znacích textu budou vyskytovat, dostaneme spolu s četností jejich výskytu ze zadání. Jestliže se znak číslo 3 vyskytuje ve 100 znacích běžného textu 4krát a v navrhovaném rozložení se vyskytuje na druhé pozici druhé klávesy, je potřeba ke každému napsání tohoto znaku zapotřebí stisknout tuto klávesu dvakrát. Do celkového počtu stisků, potřebných pro napsání 100 znaků běžného textu „přispěje“ znak číslo 3 osmi stisky ($4 * 2 = 8$). Naším cílem je tedy minimalizovat číslo, které vznikne součtem takovýchto „příspevků“, které, jak jsme si naznačili, určíme tak, že vynásobíme počet výskytů znaku ve 100 znacích běžného textu

pozici v rámci klávesy.

Asi nejprímější způsob, jak najít řešení, je projít všechny možnosti rozmístění znaků na jednotlivé klávesy, pro každé rozmístění spočítat počet stisků, potřebných pro napsání 100 znaků běžného textu s tímto rozložením a vypsát rozložení, při němž bylo dosaženo nejmenšího počtu stisků.

Jak projít všechna možná rozmístění znaků? Nejjednodušší bude, když například na první až předposlední klávesu umístíme po jednom znaku, na poslední dáme všechny zbylé. Tedy nějak takto (jednotlivé znaky si pro jednoduchost označíme symboly anglické abecedy, první jako ‚A‘, druhý jako ‚B‘, atd., až desátý jako ‚J‘):

A	B	C	DEFGHIJ
---	---	---	---------

a spočítáme potřebný počet stisků (304), poté jeden znak z poslední klávesy přesuneme na předposlední, čímž dostaneme rozložení

A	B	CD	EFGHIJ
---	---	----	--------

a počet stisků 264. Poté opět jeden znak z poslední klávesy přesuneme na předposlední a určíme, kolik stisků je potřeba, toto opakujeme až do okamžiku, kdy na poslední klávese zbude jediný znak. Jeho přesunem bychom určitě nic nezískali, proto přesuneme jeden znak ze třetí na druhou klávesu, na třetí klávese necháme jediný znak a ostatní znaky opět umístíme na poslední klávesu. Vzniklé rozložení vypadá takto:

A	BC	D	EFGHIJ
---	----	---	--------

počet stisků pro toto rozložení je 264. Obdobně budeme postupovat až do okamžiku, kdy na první, třetí a čtvrté klávese zbude po jednom znaku, všechny ostatní znaky jsou na druhé klávese:

A	BCDEFGH	I	J
---	---------	---	---

počet stisků je 188. Nyní přesuneme jeden znak z druhé klávesy na první, na druhé a třetí zůstane po jednom znaku a zbytek opět umístíme na čtvrtou klávesu:

AB	C	D	EFGHIJ
----	---	---	--------

počet stisků je 164. Tímto postupem se dostaneme až do rozložení

ABCDEFG	H	I	J
---------	---	---	---

které je konečné (počet stisků 204) a můžeme vypsát rozložení, při kterém jsme našli minimum. Je to

A	BCDE	FGHI	J
---	------	------	---

s počtem stisků 148.

Zamysleme se nyní nad tím, jak daný postup převést do programu. Program bude pracovat s poli `cetnosti` (to bude obsahovat četnosti jednotlivých znaků, načtené ze vstupu) a `delky` které bude reprezentovat právě zkoumané rozložení znaků tak, že pro každou klávesu

bude obsahovat počet znaků, které jsou na ní umístěny. Kromě toho bude program pracovat s polem `nejlepsiDelky`, kam si bude ukládat nejlepší dosud nalezené rozložení.

Na začátku se ze vstupu načtou četnosti znaků do pole `cetnosti`. Poté inicializuje pole `delky` tak, že všechny prvky kromě posledního se nastaví na 1, poslední prvek potom na $\text{počet_znaků} - \text{počet_kláves} + 1$ (tedy tak, aby součet jednotlivých prvků pole byl roven počtu znaků).

Následuje cyklus, ve kterém se vždy nejdřív určí počet stisků a pokud bylo dosaženo momentálně nejlepšího řešení, zaznamená se, poté se najde poslední klávesa, obsahující více než jeden znak. Pokud je to první klávesa (tj. všechny klávesy kromě první obsahují po jednom znaku), jsme na konci. Pokud je to poslední klávesa, jednoduše přesuneme jeden znak na předchozí (tedy o jednu zvýšíme, resp. snížíme obsah odpovídajících položek pole `delky`). V ostatních případech přesuneme jeden znak na předchozí klávesu, počet znaků na poslední klávese nastavíme na počet znaků na této klávese, počet znaků na této klávese nastavíme na 1 (promyslete si, které situaci v horním postupu to odpovídá a proč je potřeba popsání operace provést).

Poté se vypíše nejlepší nalezené rozložení spolu s počtem stisků, které vyžaduje.

K popsání zbývá zjištění počtu stisků na základě polí `cetnosti` a `delky`, ale to není myšlenkově těžké, takže ho přenechávám vaší vlastní úvaze.

Popsaný algoritmus není nejrychlejší možný, ale pro malé rozsahy bohatě postačuje. Popis složitějšího algoritmu je nad rámec tohoto textu, v zásadě je v něm využita myšlenka postupného nalezení nejlepšího rozložení pro dvě, tři a nakonec čtyři (v obecném případě n) kláves.

(jv)

2 Počítání v ZOO

2.1 Zadání

V zoologické zahradě na Cumulus Maximus mají ošetřovatelé každé ráno problémy: Psoidi, výrokanci a lemurosysi se v noci slezou do jedné ohrady a společně vyjí na měsíc. Každé ráno je potom potřeba zjistit, kolik se jich do ohrady slezlo, protože každou noc jich pár odletí, podhrabe se či uplave, nebo právě naopak.

V ohradě jsou ale namačkání tak, že není možné, jak už to v úlo-

hách tohoto typu bývá, spočítat nic jiného než nohy a hlavy. Psoidi přitom mají čtyři nohy a jednu hlavu, výrokanci tři nohy a dvě hlavy a konečně poněkud rozkolísaní lemurosylsi mají jednu nohu a tři hlavy.

Na vás je, abyste napsali program, který po zadání počtu hlav a nohou v ohradě určí, kolik kterých zvířat se může v ohradě vyskytovat, případně vypíše, že žádné řešení neexistuje.

Př.:

Vstup:

40

39

40 je počet hlav, 39 počet nohou

Výstup:

psoidů: 0, výrokanců: 11, lemurosylsů: 6

psoidů: 7, výrokanců: 0, lemurosylsů: 11

2.2 Řešení

Cílem této úlohy je najít celočíselné řešení soustavy dvou rovnic

$$4 * p + 3 * v + l = N$$

$$p + 2 * v + 3 * l = H$$

kde N a H jsou předem dané počty nohou a hlav, p , v a l jsou hledané počty psoidů, výrokanců a lemurosylsů.

První řešení je přímočaré, budeme postupně zkoušet různé kombinace p a v a budeme zjišťovat, zda pro ně lze najít l tak, že obě rovnosti jsou splněny. Kombinace p a v budeme procházet dvěma vnořenými cykly, jediná otázka je, jak určit meze těchto cyklů, tedy rozsah hodnot, pro které má smysl p a v testovat. Celkem pochopitelně je minimální zkoušená hodnota rovna nule, nejvyšší je pak dána podíly $\text{celkový_počet_nohou} / \text{počet_nohou_zvířete}$ a $\text{celkový_počet_hlav} / \text{počet_hlav_zvířete}$, resp. tím z nich, jehož hodnota je menší. Má smysl zkoumat počty zvířat pouze v tomto rozmezí (proč?) Pokud přitom určujeme meze vnořeného cyklu, je možné $\text{celkový_počet_nohou}$ i $\text{celkový_počet_hlav}$ snížit o ty, které jsou „zabrané“ vnějším cyklem.

Výsledný algoritmus by se tedy dal zapsat takto:

```
Pro p od 0 do maxPocet(celkemHlav, celkemNohou, 1, 4) {
```

```

zmenši celkemHlav o p, zmenši celkemNohou o 4*p
Pro v od 0 do maxPocet(celkemHlav, celkemNohou, 2, 3) {
    zmenši celkemHlav o 2*v, zmenši celkemNohou o 3*p
    Pokud 3*celkemHlav = celkemNohou, máš řešení
    zvyš celkemHlav o 2*v, zvyš celkemNohou o 3*p
}
zvyš celkemHlav o p, zvyš celkemNohou o 4*p
}

```

Tento program bude jistě fungovat, ukažme si ale, že pokud pro předzpracování úlohy použijeme několik matematických úprav, můžeme dosáhnout lepšího výsledku. Pokud například z první z dvojice rovnic vyjádříme l jako

$$l = N - 4 * p - 3 * v$$

a dosadíme do druhé, dostaneme

$$p + 2 * v + 3 * (N - 4 * p - 3 * v) = H$$

což se dá upravit například na

$$v = \frac{3 * N - H - 11 * p}{7}$$

dosazením do vztahu pro l dostáváme

$$l = \frac{3 * H - 2 * N + 5 * p}{7}$$

takže máme vyjádřené p i l v závislosti na v . Stačí nám tedy jediným cyklem projít všechna přípustná v a snadno zjistit, zda p a l , která dostaneme z vypočítaných vztahů jsou celá a kladná. Pokud ano, máme řešení. Druhý postup tedy vyžaduje jediný cyklus:

```

Pro p od 0 do maxPocet(celkemHlav, celkemNohou, 1, 4) {
    urči v=(3*celkemNohou - celkemHlav - 11*p) / 7
    urči l=(3*celkemHlav - 2*celkemNohou + 5*p) / 7
    Pokud jsou v i l nezáporné a celé, vypiš řešení
}

```

Celý výpočet se nám tím zjednodušil a především zrychlil. Dalšího zrychlení by šlo docílit použitím dalších matematických úprav, využívajících pravidla pro Diofantické rovnice (tak se nazývají rovnice, pro něž hledáme celočíselné kořeny).

(jv)

3 Zástup vzpěračů

3.1 Zadání

Žáci třídy se seřadili podle abecedy za sebou do jednoho zástupu. Paní učitelka pak začala hledat co nejdelší řadu žáků, kteří stojí přímo za sebou a pro něž platí, že rozdíl výšky nejvyššího a nejnižšího žáka v řadě je nejvýše 10 cm.

Pomozte paní učitelce a napište program, který dostane na vstupu počet žáků třídy a pak jejich výšky tak, jak stojí v řadě seřazení podle abecedy. Výstupem programu bude začátek a délka nejdelší souvislé řady žáků, kteří splňují uvedenou podmínku.

Př.: Vstup:

7

128

140

135

145

123

150

129

Výstup: Nejdelší řada je ze 3 žáků a začíná u 2. žáka v řadě.

3.2 Řešení

Jako obvykle jsou také u této úlohy různé způsoby řešení. Nejprve si ukážeme jednoduché, ale trochu časově náročné řešení, a pak řešení složitější, ale rychlejší.

Jednoduché řešení asi napadne po chvíli přemýšlení každého. Hledám ve vstupní řadě nějakou část řady. Tato hledaná část musí někde začínat a někde končit. Proto vyzkoušíme všechny kombinace začátků a konců. U každé takové kombinace zjistíme, zda vyhovuje zadání a z těch, které vyhovují, vybereme nejdelší. Test na splnění zadání uděláme tak, že vyzkoušíme postupně všechny dvojice výšek žáků mezi začátkem a koncem testované posloupnosti.

Postup: vstupní posloupnost výšek žáků nahrajeme do pole, abychom se mohli k výškám opakovaně vracet. Toto pole označme **VYSKY**, počet žáků, tedy velikost pole, označme **N**. Použijeme proměnnou **ZACATEK**, která bude znamenat index pole určující začátek testované kombinace. Tato proměnná **ZACATEK** může nabývat hodnot 1 až **N**. Použijeme druhou proměnnou **KONEC**, která bude znamenat index

pole určující konec testované posloupnosti. Proměnná KONEC bude nabývat hodnot ZACATEK až N.

Potřebujeme ještě další 2 proměnné (označme INDEX1 a INDEX2), které budou znamenat indexy dvojice žáků mezi začátkem a koncem, které testujeme (tedy zjišťujeme, zda se výšky liší maximálně o 10 cm). Hodnoty INDEX1 budou ZACATEK až KONEC a hodnoty INDEX2 budou INDEX1 + 1 až KONEC. Pokud se každá z dvojic výšek liší nejvýše o 10 cm, tak celá posloupnost od ZACATEK po KONEC splňuje zadání. Délka posloupnosti je KONEC – ZACATEK + 1.

Teď už jen stačí určit maximální délku — k tomu použijeme proměnnou

MAXIMDELKA, a program je hotov. Zdrojový kód programu je uveden níže.

Druhý způsob, jak již je napsáno výše, je trochu složitější. Pokud si prohlédneme řešení prvním způsobem, tak si každý jistě všimne, že je tam vnoření 4 for-cyklů v sobě. Nyní se budeme snažit se některých z nich zbavit.

Jednoduchá úvaha: Pokud posloupnost žáků splňuje podmínku, že se liší o nejvýše 10 cm, tak určitě nejnižší žák a nejvyšší žák v posloupnosti se liší nejvýše o 10 cm. Obdobně pokud podmínka splněna není, tak se určitě nejnižší žák a nejvyšší žák v posloupnosti liší o více než 10 cm. Neboli nemusíme testovat každé 2 žáky v posloupnosti, ale stačí otestovat pouze nejnižšího a nejvyššího žáka. Takto můžeme nahradit 2 vnořené for-cykly s proměnnými INDEX1 a INDEX2 jedním for-cyklem, kde najdeme minimum a maximum z výšek žáků a zjistíme, zda se liší nejvýše o 10 cm.

Druhá jednoduchá úvaha: Pokud máme posloupnost začínající na pozici ZACATEK a končící na pozici KONEC, ve které se někteří 2 žáci liší o více než 10 cm, tak určitě posloupnost, která začíná na pozici ZACATEK a končí na pozici KONEC+1, také nesplňuje podmínku zadání, protože obsahuje ty dva žáky, kteří se liší o více než 10 cm. Proto je zbytečné určovat index KONEC předem, ale je lepší postupně testovat posloupnosti od stejného začátku ZACATEK s délkami 1, 2, ... do té doby, dokud je rozdíl nejnižšího a nejvyššího žáka nejvýše 10 cm (nebo dokud nedojdeme do konce vstupní posloupnosti).

Tímto způsobem můžeme nahradit vnitřní 2 vnořené for-cykly jedním cyklem, který určuje nejdelší možnou řadu od určeného začátku. Vhodný je while-cyklus s podmínkou na nejvyšší povolený rozdíl minima a maxima.

Při hledání posloupnosti tedy budeme mít jeden for-cyklus a do něj vnořený while-cyklus. Program je opět uveden níže.

Zadání je také možné vyřešit bez použití pole se všemi výškami

žáků. Místo toho bychom potřebovali pole, kde bychom měli uložené indexy posledních výskytů hodnot, které jsou v nejdelší posloupnosti končící právě přečtenou výškou žáka. Tento způsob je sice nejrychlejší z možných, ale jeho přesný popis i přepis do programovacího jazyka je značně složitý a přesahuje složitost úloh, které se řeší na základních školách.

(pm)

4 Výpisy programů

Na dalších stránkách následují výpisy programů řešících úlohy v jazyce C.

5 Závěr

V případě, že v textu, zdrojových kódech nebo na webu najdete nějaké nedostatky či nesrovnalosti, případně budete mít nějaké dotazy, můžete se na nás obrátit prostřednictvím emailu.

Honza Vodička – Honza@Uhelnytrh.cz

Mgr. Pavel Míka – Mika@Uhelnytrh.cz