Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

# Bakalářská práce

# Průzkum a řešení ACM úloh na validačním serveru Timus

Plzeň, 2012 Petr Šiml

# Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne ..... .....

Petr Šiml

# Poděkování

Rád bych poděkoval zejména paní Ing. Arnoštce Netrvalové, PhD za její velmi přínosné rady a připomínky při vytváření této bakalářské práce.

# Abstract

**Exploring and solving ACM tasks on the validation server Timus**

The thesis deals with the academic programming contest ACM-ICPC organized by the Association for Computer Machinery and the Czech PilsProg competition for secondary schools organized by the Department of Computer Science and Engineering. It also includes the description of the ACM validation server Timus Online Judge, which contains many tasks for practise.

In the practical part of the thesis there are several tasks that were solved on the Timus Online Judge. They are applicable in the PilsProg. Each one contains Czech instructions, a description of a sufficiently efficient algorithm and files that allow us to test the accuracy of the programme. All tasks solved in the Java language are saved on a CD.

# Obsah

# Rejstřík obrázků



# Rejstřík tabulek

# 1 ÚVOD

Každým rokem stoupá počet zaměstnanců v oboru informatika, bez počítačů už si snad moderní člověk ani nedovede život představit. Kolik ale existuje opravdu dobrých pracovních míst? Jednou z možností, jak mohou studenti zvýšit svou hodnotu na trhu práce, je mít kromě specializace alespoň základní znalosti z různých odvětví. K tomu jim může výrazně pomoci příprava na některou ze soutěží v programování.

Ve své bakalářské práci bych rád připravil několik programovacích úkolů pro soutěž *PilsProg*[1], které se účastní studenti středních škol. Pokusím se vybrat co možná nejrůznorodější úlohy (co se týče postupu při jejich řešení i obtížnosti), aby byly vhodné jak pro začínající programátory, tak pro finalisty zmíněné soutěže. Jednotlivé úlohy by měly být řešitelné pomocí logiky, ale i znalosti některých základních algoritmů.

Dále k těmto úkolům vytvořím vždy určitou množinu testovacích souborů, jež bude pokrývat jak určité krajní případy, tak samozřejmě v dostatečném množství i ty běžné. Díky těmto souborům bude možné kontrolovat soutěžícím jejich programy a označit je jako správně vyřešené nebo chybné. U jednotlivých úloh vždy popíši alespoň jedno ze správných řešení.

Inspiraci při práci budu hledat zejména na serveru *Timus Online Judge*, jenž slouží k validaci ACM[2] úloh. Zmíněný server detailně popíši, aby z něho i v budoucnu mohly být čerpány návrhy na zadání pro další ročníky soutěže. Také by tento popis mohl sloužit přímo studentům středních škol, kteří by sami chtěli zdokonalovat své znalosti a hledali obdobné úlohy, jaké jsou v soutěži *PilsProg*.

[1] viz kapitola 2.2
[2] viz kapitola 2.1

# 2 TEORETICKÁ ČÁST

## 2.1 Association for Computer Machinery

*Association for Computer Machinery* (dále ACM) [5] je jednou z největších a nejznámějších společností na světě zabývajících se informatikou a výpočetní technikou v akademickém i vědeckém rozsahu. Byla založena již roku 1947, dohromady má okolo sta tisíc členů a hlavní sídlo této společnosti se nachází v New Yorku.

Společnost organizuje nejen různé konference v rozličných zemích, ale pravidelně vydává a umožňuje vydávat odborné publikace autorům z oboru. Veškeré tyto publikace jsou pak dostupné členům v elektronické podobě díky tzv. *Digital Library*. Dále ACM sponzoruje mezinárodní soutěže v programování *ACM–International Collegiate Programming Contest* [6] (dále ACM-ICPC). Ty jsou zaštiťovány některými z univerzit a slouží zejména studentům ke vzájemnému porovnání znalostí.

### *2.1.1 ACM-International Collegiate Programming Contest*

Mezinárodní soutěž v programování organizuje texaská Baylor University a její kořeny sahají až do sedmdesátých let minulého století. Od té doby prošla velkým vývojem a v dnešní době se jí každý rok pravidelně účastní tisíce týmů ze zemí celého světa. Jako každá soutěž má samozřejmě i ACM-ICPC svá organizační pravidla.

Následující pravidla jsou alespoň základním nastíněním, neboť se mohou lišit v rámci regionálních kol a platí ještě další omezení.

- V jednom týmu jsou tři vysokoškolští studenti, kteří nestudují déle než pět let a neúčastnili se více než dvakrát finále soutěže nebo pětkrát regionálního kola. Dále je pro každý rok vypsáno věkové omezení studentů.
- Každý tým má svého trenéra. Ten slouží jako prostředník mezi členy týmu a organizátory soutěže. Tým může mít jen jednoho trenéra, který ale nemůže být zároveň soutěžícím. Trenér musí oficiálně přihlásit tým do určitého data pro daný ročník.

- Všichni členové se musí zúčastnit všech soutěžních činností nebo jsou automaticky diskvalifikováni.
- Během soutěže nesmí mezi sebou týmy navzájem komunikovat. Jediný, kdo může členům něco vysvětlit či poradit, je speciálně určený personál. Ten může objasnit například některé ze systémových chyb.
- Jedno kolo soutěže trvá typicky pět hodin, počet úloh k řešení je různý, závisí na úrovni jejich obtížnosti. Minimální počet je však šest.
- Jednotlivé úkoly jsou vybírány tak, aby k jejich vyřešení nebyly nutné znalosti z některých okrajových oborů a daly se vyřešit pomocí všech dostupných jazyků (typicky C, C++ a Java).
- Dané kolo soutěže vyhrává tým, který vyřeší nejvíce úkolů. Pokud je týmů se stejným počtem úkolů více, pak jsou seřazeny podle času, během něhož je vyřešily. Pokud se nepodaří některý z úkolů odevzdat na první pokus, může na něm tým dál pracovat, nicméně při jeho úspěšném splnění je do celkového času započítáno vždy o 20 minut navíc za každý pokus o odevzdání.
- ve finálovém kole není možné používat žádné materiály kromě maximálně 25 stránek poznámek, které si studenti mohou předem připravit. Tento text je třikrát vytištěn a předložen soutěžícím před prací.

Od září do listopadu probíhají vždy nejprve regionální kola, z nichž se vítězné týmy dostávají do finále odehrávajícího se na přelomu března a dubna. Některé z týmů, které v regionálních kolech neuspěly, ale prokázaly dostatek znalostí, mohou získat ještě tzv. divokou kartu, díky níž také postoupí do finále. O správný průběh regionálních kol se stará řídící výbor *The ICPC International Steering Committee*. Řeší mimo jiné různá odvolání a drobně upravuje pravidla pro dané regiony.

ACM-ICPC klade velký důraz na týmovou práci, čemuž nasvědčuje už fakt, že (na rozdíl od většiny jiných soutěží v programování) má jeden tým jen jeden počítač. Zda je to výhoda či nevýhoda je zřejmě velmi individuální. Pro studenty je však obecně důležité, aby se naučili nejen správně a rychle řešit problémy sami za sebe, ale i umět se o své názory podělit, rozhodnout, který z návrhů je ten nejlepší nebo si jednotlivé úlohy na začátku rozdělit mezi všechny členy týmu.

Česká republika se také zapojuje do soutěže v regionálním kole *Central European Regional Contest* [7].

## 2.2 PilsProg

*PilsProg* [8] je soutěž v programování, která je (narozdíl od ACM-ICPC) zaměřena na porovnávání schopností středoškolských studentů. Organizátorem soutěže je Katedra informatiky a výpočetní techniky (KIV) Fakulty aplikovaných věd na Západočeské univerzitě v Plzni. Letošní ročník PilsProg je již pátým v historii soutěže.

Princip fungování a obecně pravidla jsou v lecčems podobná zmíněné soutěži organizované společností ACM, přece se však v něčem liší. Nejprve probíhají tzv. kvalifikační kola, kde jednotliví studenti (nejsou zde žádné týmy) sami po úspěšné registraci odevzdávají řešení úloh přes oficiální web soutěže. Po skončení této části jsou vybrání nejlepší studenti postupující rovnou do finále, které se koná přímo v prostorách KIV.

### 2.2.1 Řešení úloh

PilsProg umožňuje studentům programovat v jazycích Java, C a Pascal bez používání některých knihoven, např. pro otevírání souborů. Není možné zahrnovat do svých kódů diakritiku a to ani v komentářích. Výsledným řešením se rozumí jeden zdrojový soubor daného programovacího jazyka, který načítá data ze standardního vstupu (klávesnice), vypisuje výsledky na standardní výstup (obrazovka) a úspěšně projde přes vyhodnocovací systém soutěže. Ten testuje program pro několik předem připravených souborů se správnými výstupními hodnotami pro různé vstupy.

### 2.2.2 Rozdíly soutěží

Oproti ACM-ICPC v kvalifikačních kolech neplatí pravidlo o časové penalizaci při úspěšném odevzdání úlohy na jiný než první pokus. Nicméně ve finále je toto pravidlo opět zahrnuto – s penalizací 10 minut pro každé nesprávné odevzdání úlohy. Dále je zde rozdíl v dostupných materiálech. V soutěži PilsProg si mohou studenti do finále vzít libovolné množství knih či textu.

### 2.2.3 Validační zprávy

Aby studenti rozpoznali, zda jimi vytvořený program úspěšně prošel všemi testy či nikoli, vrací validační systém několik typů zpráv. Soutěžící se tak mohou dozvědět, že jejich program neprošel kvůli chybě při kompilaci, překročenému časovému limitu, práci s nepovolenou knihovnou či chybnému řešení.

## 2.3 On-line validátory

Existuje mnoho on-line validačních serverů, přičemž ne vždy je úplně jasné, zda se jedná o samostatné projekty nebo jsou primárně vytvořeny za účelem přípravy pro soutěž ACM-ICPC. Určitě by neuškodila větší provázanost těchto stránek, kdy by stačilo doplnit byť strohý seznam spřátelených serverů společně s odkazy na jejich domovské stránky, případně tento seznam umístit přímo na web ACM. Studenti by to jistě ocenili. Jediným místem, kde je uvedeno hned několik odkazů na jednotlivé validační systémy jsou webové stránky [9].

Postupně jsem se registroval na *UVA Online Judge* [10], *Sphere Online Judge* (*SPOJ*) [11] a *Timus Online Judge* [12] abych mohl tyto servery vzájemně porovnat. Princip fungování je na všech stejný – po registraci lze odevzdávat řešení jednotlivých úloh a systém prakticky ihned odpoví, zda je testovaný program správně, a pokud ne, upozorní např. na chyby při překladu či oznámí, že na některý z testovacích vstupů neodpověděl program správně.

Úkolů jsou zde opravdu stovky a nejvíce se servery liší v možných programovacích jazycích, kde zejména na serveru SPOJ lze odevzdávat opravdu téměř v jakémkoli jazyce, třeba v assembleru. Na serveru UVA bohužel chybí diskuze k jednotlivým úlohám, které se velmi hodí, protože v nich uživatelé často píší, na jakých vstupních datech jejich programy neuspěly či nastíní některé z možných řešení daného problému.

## 2.4 Timus Online Judge

### 2.4.1 Základní informace

*Timus Online Judge* (dále jen *Timus*) je jedním z ACM serverů, na němž mohou programátoři z celého světa testovat své znalosti z oboru. Funguje 24 hodin denně a jeho spravování spadá pod Ural State University. Hlavním jazykem by se tedy nabízela být ruština, nicméně autoři se rozhodli jako základní zvolit angličtinu. Toto omezení platí i pro diskuze k jednotlivým úlohám, bohužel ne vždy a všude se dodržuje.

Celý web je velmi přehledný i pro začínající angličtináře a díky horní liště se lze navigovat rovnou na potřebné stránky (viz Obrázek 1).


**Obrázek 1 - Horní lišta webových stránek serveru Timus**

### 2.4.2 Seznámení a registrace

Na serveru je dohromady necelých 900 úloh k programování, ovšem pro testování správných řešení je nutné se nejprve registrovat. Na zadaný email pak přijde nejen potvrzení o registraci, ale i velmi důležité identifikační číslo. Pod zvolenou přezdívkou („Name“) lze nalézt jednotlivé řešitele úloh například v diskuzích či v žebříčku nejlepších programátorů, ale pro samotné kontrolování vypracované úlohy nebo přihlašování se používá jen přiřazené ID. Při registraci je nutné zadat i zemi původu, která samozřejmě nehraje žádnou roli v řešení úloh, nicméně je zajímavé sledovat, jak si v žebříčku nejlepších programátorů vedou jaké státy.

### 2.4.3 Sada úloh

Timus byl stvořen pro zlepšování a porovnávání umu programátorů, proto nejdůležitější částí stránek je zřejmě sada testovacích úloh. Ty jsou rozděleny do několika kategorií – jsou zde zadání jak pro dynamické programování, tak třeba pro geometrické úlohy, přičemž všechny vyžadují do určité míry originální způsob řešení

[1, 2, 3]. Každý problém má také své unikátní číslo a je možné nehledět jen na kategorizaci, ale postupovat s řešením jedné úlohy po druhé.

Při zobrazení zvoleného seznamu (viz Obrázek 2) s ním lze jednoduše pracovat. Je možné schovat již vyřešené úlohy (což se hodí spíše až ve fázi, kdy je jich více vyřešených než nevyřešených) a hlavně všechny řadit. Základní prioritou pro seřazení je identifikační číslo úlohy, ale lze zobrazovat postupně i podle počtu uživatelů, kteří úlohu vyřešili, či podle obtížnosti. Nikde na stránkách Timusu není uvedeno, jakým způsobem se tato obtížnost počítá, nicméně pravděpodobně zobecňuje procentuální úspěšnost a neúspěšnost řešitelů. Mnohokrát ale není řazení podle obtížnosti příliš relevantní.

**Problem set**

Solved problems: show | hide • Sort by: id | authors | difficulty

| | ID | Title | Source | Authors | Difficulty |
|---|---|---|---|---|---|
| ✓ | 1000 | A+B Problem | | 45304 | 17 |
| | 1001 | Reverse Root | | 10324 | 26 |
| | 1002 | Phone Numbers | Central European Olympiad in Informatics 1999 | 4192 | 168 |
| | 1003 | Parity | Central European Olympiad in Informatics 1999 | 2286 | 307 |
| | 1004 | Sightseeing Trip | Central European Olympiad in Informatics 1999 | 2703 | 260 |
| | 1005 | Stone Pile | USU Championship 1997 | 9910 | 71 |
| | 1006 | Square Frames | USU Championship 1997 | 1125 | 620 |
| | 1007 | Code Words | USU Championship 1997 | 2676 | 262 |
| | 1008 | Image Encoding | USTU PhysTech Cup 2000 | 2977 | 236 |
| | 1009 | K-based Numbers | USU Championship 1997 | 8628 | 81 |
| | 1010 | Discrete Function | USTU PhysTech Cup 2000 | 3800 | 185 |

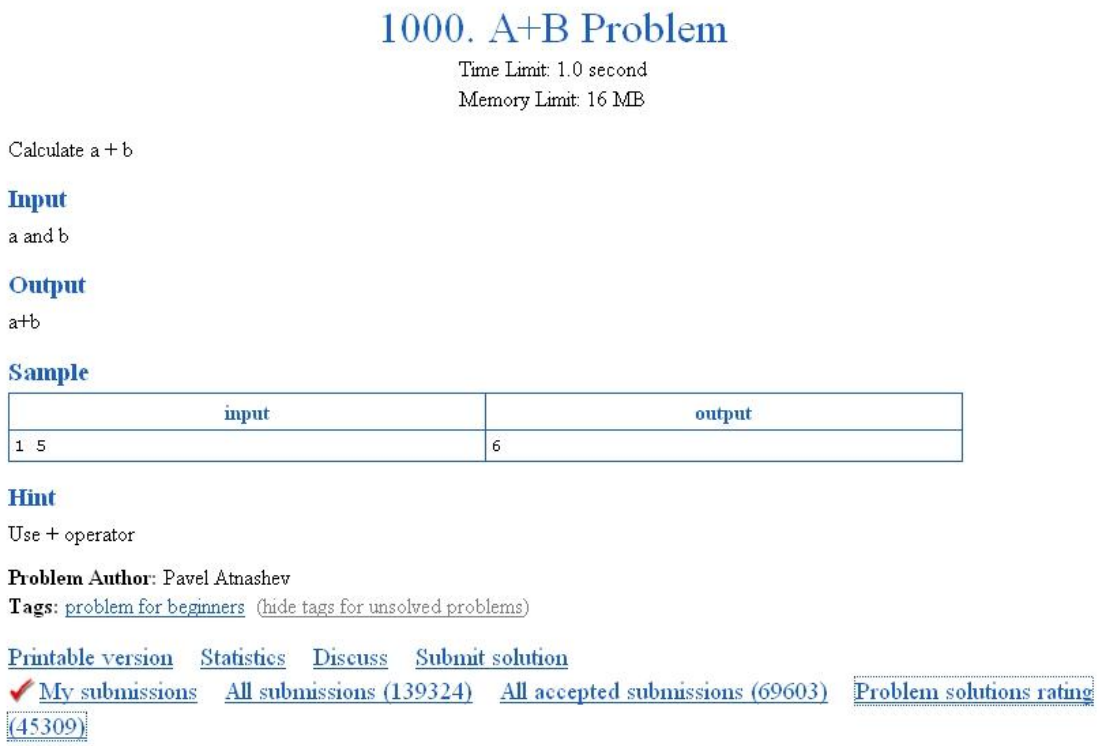
**Obrázek 2 – Ukázka seznamu úloh na serveru Timus**

### 2.4.4 Jednotlivé úlohy

Po kliknutí na zvolený úkol se zobrazí jeho zadání spolu s několika dalšími údaji (viz Obrázek 3). Hned pod titulkem jsou dvě důležité informace – časový a paměťový limit. Při validování je použita množina vstupních a výstupních dat. Správné odpovědi ovšem nestačí, protože program nesmí překročit jak určitý čas, tak paměť (od začátku prvního vstupu do skončení po posledním). Díky tomu jsou programátoři nuceni neustále vymýšlet nové a zlepšovat již použité algoritmy.

Pro danou úlohu je zde vždy malá nápověda v podobě jednoho z nejjednodušších vstupů a k němu správného výstupu. Všichni si tak mohou zkontrolovat, že správně porozuměli zadání. Občas se bohužel stane, že některá zadání mají drobné odlišnosti

v ruštině a angličtině, což může někdy hrát velkou roli. Naštěstí je to většinou zmíněno někým v diskuzi a problém je následně vyřešen administrátory.



1000. A+B Problem

Time Limit: 1.0 second
Memory Limit: 16 MB

Calculate a + b

**Input**

a and b

**Output**

a+b

**Sample**

| input | output |
|---|---|
| 1 5 | 6 |

**Hint**

Use + operator

**Problem Author**: Pavel Atnashev
**Tags**: problem for beginners (hide tags for unsolved problems)

Printable version Statistics Discuss Submit solution
My submissions All submissions (139324) All accepted submissions (69603) Problem solutions rating (45309)

**Obrázek 3 - Zadání jedné z úloh**

Právě diskuze občas bývá velmi důležitým prvkem. Každý se zde může s ostatními podělit o své zkušenosti či se zeptat na něco konkrétního, např. jaký je výstup pro určitý konkrétní vstup. Není dovoleno uveřejňovat své již zvalidované zdrojové kódy, ale to je jen dobře, protože jinak by celý tento systém neměl smysl. Je trochu škoda, že není lépe vyřešeno odpovídání na dotazy v diskuzích, protože není ničím neobvyklým, že někteří uživatelé se dozvědí odpověď až po několika měsících nebo také vůbec.

U každé úlohy jsou vedeny statistiky. Lze tak zjistit, kolik z odevzdaných programů úspěšně prošlo kontrolou, či jaký uživatel zvládl napsat program řešící tento problém v nejkratším čase. Komunikace mezi uživateli ale není možná, protože kromě přezdívky a země původu není uveřejněno o uživatelích nic osobního, ani registrační email. Jedinou možností, jak pro zajímavost zjistit nejrychlejší možný algoritmus, je tak napsat do diskuze.

### 2.4.5 Odevzdávání

Pro řešení úloh na Timusu je možné využít programovací jazyky Java, Pascal, C, C++ a C#. Po vypracování programu v jednom z těchto jazyků je při odevzdání úlohy nutné daný jazyk nastavit, aby se použil správný kompilátor. Zdrojový kód je možné jak nakopírovat přímo do textového pole na stránce, tak vybrat soubor uložený v počítači.

Validátor po otestování programu během chvíle zobrazí svou odpověď. Možnosti jsou vypsány v Tabulce 1.

| | |
|---|---|
| Accepted | Jediná odpověď označující, že vše proběhlo, jak mělo, a program splnil nejen časový a paměťový limit, ale i odpověděl na všechny testy správně. |
| Compilation error | Nastal problém při kompilaci. V programu je chyba typu překlepu v názvu proměnné nebo volání neexistující metody. |
| Wrong answer | Program se sice správně přeložil, ale na některý z testů neodpověděl správně. Je zobrazeno i číslo pořadí testovacího vstupu, na kterém neuspěl. |
| Time limit | Byl překročen časový limit určený v zadání. Je možné, že uvnitř programu je nekonečný cyklus nebo že zvolený algoritmus není dostatečně efektivní. |
| Memory limit | Byl překročen paměťový limit určený v zadání. |
| Output limit | Program začal vypisovat více věcí než je potřeba pro splnění úlohy. |
| Crash | Nastala výjimka při běhu programu. Může ji způsobit např. dělení nulou nebo přístup k buňce pole s větším číslem, než je jeho délka. |

**Tabulka 1 - Odpovědi serveru po kontrole programu**

Je určitě velkou výhodou, že ať už je odevzdávaný program správně či nikoli, uloží se na serveru a lze k němu pak vždy přistupovat po přihlášení. Není tedy nutné zálohovat si pokaždé své zdrojové kódy.

### 2.4.6 Osobní stránka

Po vyřešení úlohy se pro uživatele označí jako úspěšně splněná a z jeho osobní stránky je pak přehledně vidět, kolik úloh má již hotových a o které se zatím neúspěšně pokoušel. Dále je zde uvedeno jeho umístění v celkovém žebříčku všech řešitelů. Žebříček je sestaven, možná trochu nespravedlivě, podle počtu odevzdaných úloh bez ohledu na jejich obtížnost.

### 2.4.7 Začátky

Pro důvěrnější seznámení se systémem odevzdávání úloh je dobré začít tou úplně první, v níž se jen sečítají dvě čísla. Tento základní problém samozřejmě není těžké vyřešit, ale občas se s něčím vyskytne potíž. Například pro zdrojové kódy v jazyce Java nelze odevzdávat s přiřazením programu do balíku (package). Pokud tedy program používá více tříd, musí být všechny v jednom souboru.

# 3 PRAKTICKÁ ČÁST

## 3.1 Vybrané příklady a jejich struktura

Pro příklady, které jsem na serveru Timus vybral a přepracoval do podoby pro soutěž PilsProg, jsem vytvořil české zadání, různé množství testovacích souborů a popsal použitá řešení. Vše je umístěno na přiloženém CD včetně originálního zadání pro případné nejasnosti a zdrojových kódů v jazyce Java [4]. Následujících několik úloh je pouze podmnožinou reprezentující různé postupy při řešení.

U každého úkolu je v závorce uvedeno číslo originálního zadání na serveru Timus.

## 3.2 Binární řetězce (1081)

### Zadání

Mějme všechny řetězce skládající se ze znaků 1 a 0 s délkou $0 < N < 44$, přičemž dvě jedničky nemohou být v řetězci umístěny za sebou (110 není platný řetězec délky 3, 0101 je platný řetězec délky 4). Napište program, který najde řetězec, jenž je na $K$-tém místě ($0 < K < 10^9$) v lexikograficky vzestupně seřazených řetězcích.

Poznámka: 00 < 01 < 10

### Vstup

$N$, $K$ - celá čísla oddělená mezerou.

### Výstup

Nalezený řetězec nebo -1, pokud číslo $K$ je větší než počet řetězců splňujících pravidla pro danou délku.

### Příklad

vstup:

3 1

výstup:

000

## Řešení

Ze zadání je zřejmé, že pokud platí např. $N = 2$, pak existují tři možnosti 00, 01 a 10. 11 není platný řetězec. Aby bylo možné vypsat správný řetězec, musí být pro tento příklad splněna podmínka $K < 4$.

Pro obecné určení, zda číslo $K$ není moc veliké, tj. zda vůbec existuje takové množství správných řetězců délky $N$, vytvoříme pole, jehož první dvě hodnoty nastavíme na 1 (jen teoreticky pro délku řetězců 0) a 2 (pro délku 1). Pro zvolenou délku $N$ totiž platí zajímavá závislost. Sečtením dvou předchozích hodnot (čili počtu prvků pro délku $N$-1, $N$-2) zjistíme počet prvků pro délku $N$ - platí zde pravidlo pro Fibonacciho posloupnost [13]. Postupným sčítáním tak zjistíme počty řetězců pro všechny délky $L \leq N$ (viz Tabulka 2 pro délku $N = 10$).

| Počet řetězců | 2 | 3 | 5 | 8 | 13 | 21 | 34 | 55 | 89 | 144 |
|---|---|---|---|---|---|---|---|---|---|---|
| Délka | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

**Tabulka 2 - Počet řetězců pro délky 1-10**

Po zjištění, že $K$-tý řetězec opravdu existuje, ho musíme najít. Využijeme při tom již vypočítané pole s maximálním počtem řetězců pro různé délky. Platí-li totiž, že pro počet prvků délky $i$-1 je $K$ větší, bude na ($N$+1-$i$)-té pozici řetězce číslo 1. Od $K$ pak odečteme hodnotu pole s indexem $i$-1 a počítáme dále.

Ukažme si algoritmus na příkladu pro $N = 8$, $K = 54$:

Vytvoříme pole o délce 9, kde na indexu 0 bude hodnota 1, na zbylých indexech budou hodnoty z Tabulky 2. Následně v cyklu procházíme sestupně toto pole od indexu $i = 8$. Hned pro index 7 (tedy $i$-1) platí podmínka *počet řetězců* < $K$ (34 < 54), tedy první znak řetězce bude 1. Nastavíme $K = 54 - 34 = 20$ a pokračujeme v cyklu. Dále platí:

- jelikož není splněna podmínka 21 < 20, druhým znakem řetězce bude 0.
- podmínka 13 < 20 už splněna je, tedy třetím znakem bude 1 a $K = 20 - 13 = 7$.
- podmínka 8 < 7 opět neplatí. V tuto chvíli máme výsledný řetězec "1010".
- protože 5 < 7, pátý znak bude opět 1 a $K = 7 - 5 = 2$.
- 3 < 2 ani 2 < 2 neplatí, na dalších dvou pozicích tedy budou znaky 0.

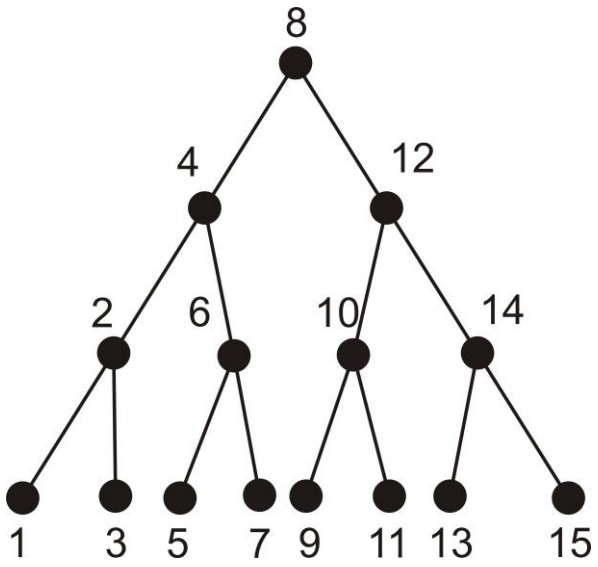
- protože máme na nultém indexu pole hodnotu 1, tato podmínka je splněna posledním znakem řetězce bude 1.

Výsledný řetězec je tedy "`10101001`".

## 3.3 Doručení zpráv (1117)

### Úvod

Během dlouhých let vznikla ve firmě Tuneláři a.s. určitá hierarchie zaměstnanců. Každý pracovník (kromě těch na nejnižších pozicích) má právě dva přímé podřízené a každý (kromě ředitele firmy) má jednoho přímého nadřízeného. Dále má každý ze zaměstnanců své unikátní číslo, přičemž nejvyšší číslo se rovná počtu zaměstnanců a nejnižší je číslo 1. Počet úrovní mezi nejvýše postaveným ředitelem a zaměstnanci na nejnižší pozici je pro všechny tyto zaměstnance stejný. Struktura číslování zaměstnanců je taková, že každý nadřízený, má větší číslo než jeden z jeho podřízených a menší číslo než druhý. Dále platí, že pokud má podřízený vyšší (nižší) číslo než jeho šéf, pak i jeho podřízení mají vyšší (nižší) čísla, než má tento šéf (viz Obrázek 4 pro 15 zaměstnanců).




**Obrázek 4 – Ukázka hierarchie pro 15 zaměstnanců**

### Zadání

Byl vypracován zvláštní systém předávání zpráv mezi zaměstnanci. Zpráva od zaměstnance s číslem *i* může být přímo doručena jen zaměstnanci *i*+1 nebo *i*-1. To se děje v průběhu jednoho dne (trvá to 0 dnů), pokud vztah mezi pracovníky s těmito čísly je přímý, bez prostředníka. Jinak doručení zprávy trvá tolik dnů, kolik je prostředníků

pro její doručení. Vaším úkolem je sepsat program, který určí počet dní pro doručení zprávy od odesílatele k příjemci.

Například: Pracovník s číslem 2 posílá zprávu pracovníkovi číslo 4. Nejprve ji pošle zaměstnanci číslo 3 a ten až ji odešle číslu 4. Proces trvá 1 den, protože první krok (2→3) trvá 0 dní, druhý krok (3→4) 1 den.

## Vstup

Dvě čísla zaměstnanců oddělená mezerou, první číslo označuje odesilatele zprávy a druhé příjemce. Počet zaměstnanců není větší než $2^{31}$-1.

## Výstup

Počet dní, během nichž dorazí zpráva k příjemci.

## Příklad

vstup:
1 5

výstup:
2

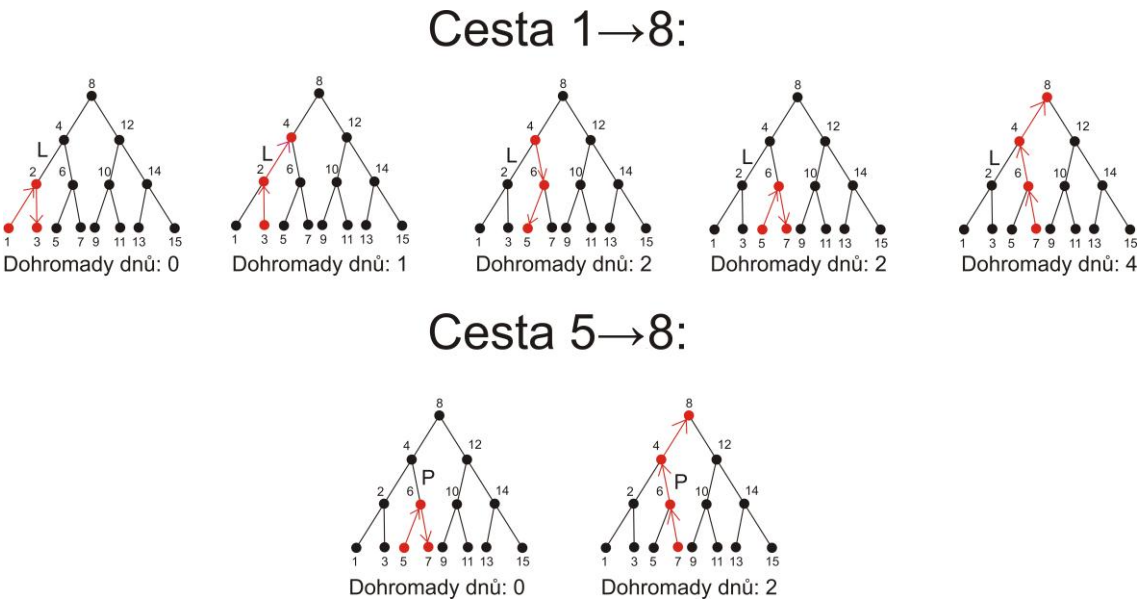
## Řešení

Před řešením je dobré si uvědomit, že velikost stromu se všemi zaměstnanci lze zmenšit tím, že odesílatele a příjemce zprávy přesuneme na levou stranu z pravé části (pokud tam již nejsou rovnou). Předání zpráv mezi zaměstnanci 10→11 z obrázku výše se tak může přenést na předávání zpráv ve stromu mezi zaměstnanci 2→3 atd..

Po načtení obou čísel tedy měníme čísla zaměstnanců tak, aby se strom co nejvíce zmenšil a počet dní předání zprávy zůstal stejný. Dále uvažujeme jen vzniklý podstrom. V další fázi počítáme pro každého zaměstnance zvlášť, nicméně stejným způsobem, jen druhého zaměstnance symetricky (podle osy) v rámci podstromu překlopíme do levé větve.

Pro zjištění počtu dnů při předávání zprávy nám stačí určit pro každého zaměstnance cestu od nejvyššího nadřízeného. Např. pokud uvažujeme na Obrázku 4 cestu 1→11, označíme nejvyššího šéfa obou zaměstnanců jako číslo 8. Jelikož počítáme pro oba zaměstnance zvlášť a číslo 11 lze překlopit doleva tak, aby byl počet dnů pro přenos k nadřízenému zachován (uvažujeme ho jako číslo 5), víme, že první cesta povede vždy doleva – tedy ji neuvažujeme. „Zástupcem“ nejvyššího nadřízeného se tedy stává číslo 4. Od něho vede k prvnímu zaměstnanci cesta doleva – zapíšeme ji jako ‘L‘ do řetězce. Podobným způsobem pokračujeme dále, nacházíme cestu a v daném podstromu zapisujeme do řetězce vždy ‘L‘ jako doleva či ‘P‘ jako doprava. Na nejnižší úrovni není nutné stranu určovat, jelikož předání zpráv trvá 0 dní. V tomto případě tedy v řetězci zůstane právě jen ‘L‘ (pro číslo 5 je v řetězci ‘P‘).

Po vytvoření řetězce lze snadno spočítat, za kolik dní se zpráva dostane do nejvyššího čísla levé větve stromu, v tomto případě čísla 7. Pokud je totiž v řetězci znak ‘L‘, k výsledku připočítáme počet dnů předání zpráv od nejvyššího čísla v levé větvi podstromu k nejvyššímu číslu ve větvi pravé, přičemž postupně uvažujeme stále menší a menší podstromy. Od čísla 7 se pak zpráva k nejvyššímu řediteli dostane za počet dnů, kolik mezi nimi leží pater stromu. Postup konkrétněji viz Obrázek 5.




**Obrázek 5 - Cesty od zaměstnanců 1, 5 k nadřízenému č. 8**

Drobný problém nastává, pokud zprávu neposílá řadový zaměstnanec, ale některý z nadřízených. Musíme nejprve spočítat, kolik pater stromu od něho vede k o číslo většímu zaměstnanci – jelikož po sudém čísle následuje liché, jedná se vždy o zaměstnance bez podřízených.

Po určení počtu dnů pro předání zprávy od obou zaměstnanců k nejvyššímu nadřízenému získáme celkový výsledek sečtením obou hodnot. Pro uvedený příklad 1→11 je tedy výsledek 4 + 2 = 6.

## 3.4 Hledání hesla (1547)

### Zadání

Po prázdninách zapomněl František své heslo ke školnímu emailu. Naštěstí má ve své škole přístup k výkonnému počítači a může tak použít *M* procesorů k nalezení svého hesla. Franta si pamatuje, že jeho heslo není delší než *N* znaků a skládá se jen z malých písmen anglické abecedy. Nejprve chce otestovat všechny možnosti pro délku 1 (a, b...z), poté pro délku 2 (aa, ab... zz) atd.

Aby bylo hledání efektivní, musí se množina rozdělit rovnoměrně mezi všechny procesory. První část bude prohledávat první procesor, druhou druhý.. a *M*-tou *M*-tý. Pokud není možné rozdělit práci rovnoměrně, prvních *x* procesorů bude kontrolovat vždy o 1 slovo více než ostatní. Franta by chtěl vědět, který procesor bude testovat jaké řetězce.

### Vstup

Jedna řádka obsahující dvě čísla *N*, *M* oddělená mezerou. Počet zkoumaných slov není menší než počet procesorů.

### Výstup

*M* řádek, v *i*-té řádce bude vypsán rozsah slov pro *i*-tý procesor (viz příklad).

### Příklad

vstup:

5 4

výstup:

a-fssst

fsssu-mmmmn

mmmmo-tgggg

tgggh-zzzzz

## Řešení

Řetězce jsou složeny z písmen, které tvoří v podstatě vlastní číselnou soustavu (od ‘a‘ do ‘z‘). Jelikož se jedná jen o znaky z anglické abecedy, můžeme pro tyto znaky použít ASCII tabulku, kde jednoduše zjistíme, že jich je 26, přičemž kód znaku ‘a‘ je 97 a znaku ‘z‘ 122.

Postupujeme pak velmi jednoduše. Nejprve si spočítáme počet všech slov, která musí procesory zkontrolovat – pro délku 1 je to 26, počet slov délky 2 je $26^2$... až pro délku $N$ $26^N$. Počet slov ale nemusí být rovnoměrně rozdělený mezi všechny procesory, proto ještě výsledek vydělíme modulo počtem procesorů a zjistíme tak, kolik prvních bude kontrolovat o jedno slovo více.

Nyní nám stačí zjistit algoritmus, který z konkrétního čísla vytvoří jemu odpovídající řetězec. Pro snazší pochopení si ukažme známý postup převodu čísla z desítkové soustavy do šestnáctkové:

Mějme číslo $(2012)_{10}$, pro jeho reprezentaci v šestnáctkové soustavě je možné provést následující postup:

1. 2012 modulo 16 = 12, což je reprezentace znaku C
2. 2012 – 12 = 2000.
3. 2000 / 16 = 125.
4. 125 modulo 16 = 13, což je reprezentace znaku D
5. 125 – 13 = 112.
6. 112 / 16 = 7
7. platí 7 < 16, tudíž převod ukončíme. Číslo 7 je stejné v desítkové i šestnáctkové soustavě.

Číslo 2012 vyjádřené v hexadecimální soustavě je tedy $(7DC)_{16}$.

Podobně pracujeme i s převodem do “naší dvacetišestkové“ soustavy. Vyskytuje se tu ovšem problém, protože tato soustava nemá žádný znak, který by nahradil nulu. Začíná znakem ‘a‘ a po znaku ‘z‘ následuje rovnou řetězec “aa“, čili neexistuje něco jako číslo 10 v desítkové soustavě, ale rovnou číslo 11. Pokud je zbytek po celočíselném dělení 26ti různý od 0, pracujeme stejně jako v ukázkovém příkladu pro převod a každé číslo reprezentuje jedno z písmen. Pokud je ovšem zbytek roven 0, jedná se o znak ‘z‘ a po vydělení čísla 26ti ještě musíme odečíst 1.

Ukázka postupu pro převod čísla $(2340)_{10}$:

1. 2340 modulo 26 = 0, tedy znak ‘z‘
2. 2340 / 26 = 90.
   - jelikož byl zbytek roven 0, provedeme ještě: 90 – 1 = 89
3. 89 modulo 26 = 11, jedenáctým znakem je ‘k‘
4. 89 – 11 = 78
5. 78 / 26 = 3
6. platí 3 < 26, převod ukončíme. Třetím znakem abecedy je ‘c‘.

Výsledný řetězec je “ckz“.

## 3.5 Počítání pater (1564)

### Zadání

Petr rád chodí po schodech až na vrchol mrakodrapů. Naposledy ale nedospěl až do nejvyššího patra, protože zakopl a upadl do bezvědomí. V nemocnici se pak dokázal rozvzpomenout jen na celkový počet číslic 1 v číslech poschodí, které cestou minul. Pomozte Petrovi zjistit, v jakém poschodí upadl.

### Vstup

Počet jedniček, který Petr napočítal (číslo v rozmezí 1 až $10^{18}$).

### Výstup

Číslo poschodí, v kterém Petr upadl. Pokud je číslo nesprávně zadáno (např. pro číslo 3 neexistuje žádné číslo poschodí, protože po patře s č. 10 následuje č. 11, čili z počtu

jedniček 2 se přejde rovnou na 4), vypište -1. Pokud je číslo zadáno správně, vypište nejmenší možný výsledek (pro číslo 13 je správným výstupem 21, ne např. 25).

### Příklad

vstup:
4

výstup:
11

### Řešení

Při řešení tohoto příkladu využijeme známého algoritmu binárního vyhledávání. Na začátku nastavíme hodnotu pro binární vyhledávání (označme jako *binarni*) na $2^{62}$, dvojnásobek tohoto čísla výsledek nikdy nepřeroste. Na stejnou hodnotu nastavíme i číslo obsahující interval dělení (označme jako *interval*).

Dále pracujeme v cyklu následovně:

- zjistíme si, kolik jedniček dohromady obsahují všechna čísla menší než *binarni* (včetně čísla samotného).
- pokud je tato hodnota větší než číslo ze vstupu, vydělíme *interval* dvěma a přiřadíme *binarni* = *binarni* – *interval*
- pokud je tato hodnota menší než číslo ze vstupu, vydělíme interval dvěma a přiřadíme *binarni* = *binarni* + *interval*
- pokud je tato hodnota stejná jako číslo ze vstupu, víme, že výsledné číslo existuje. Abychom ho našli, musíme z čísla *binarni* odečítat jedničky dokud tato podmínka platí.
- číslo neexistuje, pokud se interval dělením dostane až k nule.

Pro zjištění počtu jedniček *pocet* pro určité číslo *cislo* na začátku přiřadíme

*pomCislo* = *cislo*

Pak se ve `for` cyklu (od hodnoty *i* = 0) vždy zjistí zbytek po celočíselném dělení *pomCislo* deseti, odečte se od *pomCislo* a to se pak vydělí deseti. Výsledný počet jedniček počítáme následovně:

- pokud byl zbytek 0:

$$pocet \mathrel{+}= pomCislo \cdot 10^{i};$$

- pokud byl zbytek 1:

$$pocet \mathrel{+}= pomCislo \cdot 10^{i};$$
$$pocet \mathrel{+}= (cislo \bmod 10^{i}) + 1;$$

- pokud byl zbytek větší než 1:

$$pocet \mathrel{+}= (pomCislo + 1) \cdot 10^{i};$$

Pro názornost si ukážeme výpočet množství jedniček pro číslo 1015:

1. 1015 modulo 10 = 5, tedy $pocet \mathrel{+}= (101 + 1) \cdot 10^{0} = 0 + 102 = 102$
2. 101 modulo 10 = 1, tedy $pocet \mathrel{+}= 10 \cdot 10^{1} = 102 + 100 = 202$
   $pocet \mathrel{+}= (1015 \,\%\, 101) + 1 = 202 + 5 + 1 = 208$
3. 10 modulo 10 = 0, tedy $pocet \mathrel{+}= 1 \cdot 10^{2} = 208 + 100 = 308$
4. 1 modulo 10 = 1, tedy $pocet \mathrel{+}= 0 \cdot 10^{3} = 308 + 0 = 308$
   $pocet \mathrel{+}= (1015 \,\%\, 103) + 1 = 308 + 15 + 1 =$ **324**

## 3.6 Barevné kostky (1127)

### Zadání

Ve školce je mnoho velkých barevných kostek, z nichž děti stavějí věž. Ta má jako základnu jen jednu kostku a vždy, když ji děti postaví, zase ji ihned zboří. Jednotlivé kostky mají každou stranu jinak barevnou a učitelky ve školce nutí děti k tomu, aby věž měla vždy stejně barevné všechny čtyři stěny. Jelikož čím větší je věž, tím lépe se boří, děti ji chtějí postavit co možná nejvyšší.

### Vstup

První řádka obsahuje číslo $N$ ($1 < N \leq 1000$), počet kostek. Na dalších $N$ řádcích je pak vždy popis jedné kostky - řetězec šesti písmen popisujících barvu dané strany v pořadí: přední, pravá, levá, zadní, horní, dolní.

Barvy jsou následující: *A* - azurová, *M* - modrá, *Z* - zelená, *O* - oranžová, *R* - rudá, *F* - fialová, *B* - bílá, *H* - hnědá, *S* - stříbrná, *L* - levandulová. Kostka nemá nikdy dvě stejně barevné stěny.

## Výstup

Jediné číslo - maximální výška věže podle daných pravidel.

## Příklad

vstup:
4
ZHFABM
AORZHF
RABFZO
OFHMZA

výstup:
3

## Řešení

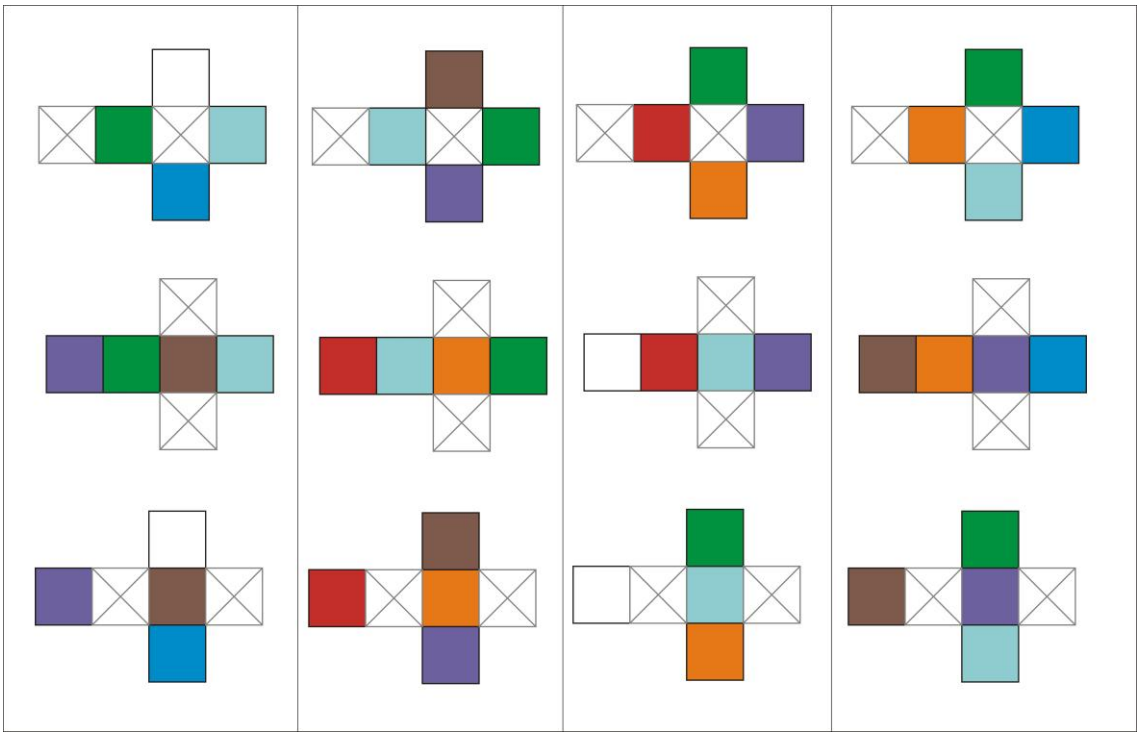
Nejprve je dobré si uvědomit, kolika způsoby lze jednu kostku položit. Nestačí nám jen spočítat, na kolika stranách může kostka ležet, protože každé její natočení nám dává novou možnost. V tomto příkladu nám jde ovšem jen o 4 sousední strany, nezáleží na tom, jaká barva kostky je dole a jaká nahoře. Seřadíme-li tedy tyto čtyři strany vždy stejným způsobem, možnosti pro jednu kostku jsou jen tři. Konkrétněji viz následující případ pro první kostku ze vzorového příkladu (kostky jsou znázorněny na Obrázku 6):

ZHFABM – zelená, hnědá, fialová, azurová, bílá a modrá. Jaké čtyři strany můžeme pro stavbu věže započítat?

- přední, levou, zadní a pravou
- horní, levou, dolní a pravou
- horní, přední, dolní a zadní

Pro každou z možností navíc existuje osm různých pořadí. Vezmeme-li ale každou barvu reprezentovanou jedním znakem jako jeho hodnotu v ASCII tabulce, můžeme je vzestupně seřadit tak, aby byla posloupnost barev jednoznačná. Pro ZHFABM tedy:

- AFZH
- BFMH
- ABZM



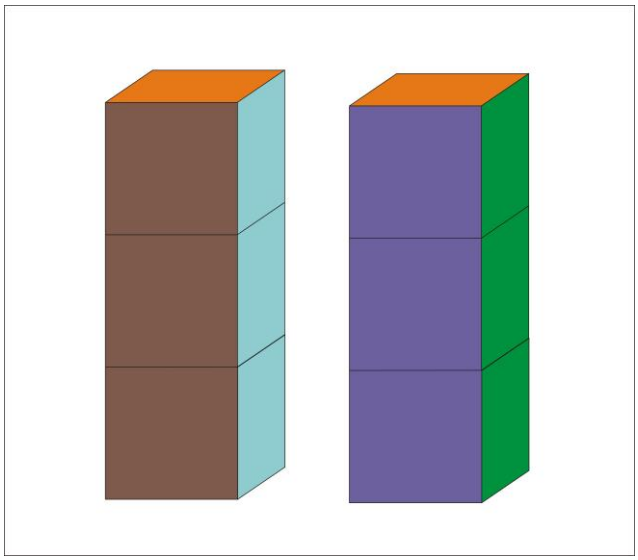
**Obrázek 6 - Použitelné kombinace kostek ze zadání**

Při této znalosti můžeme pro každou kostku počítat jen tři posloupnosti a můžeme všechny uchovávat v paměti včetně čísla znázorňujícího počet, kolik kostek stejnou posloupnost obsahuje (viz Tabulka 3).

| Posloupnost | AOBZ | FORZ | AMZO | FOHR | AFBR | BFMH | AOZR | AFZH | ABZM | FMHO |
|---|---|---|---|---|---|---|---|---|---|---|
| Kolik kostek ji obsahuje | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 3 | 1 | 1 |

**Tabulka 3 - Všechny seřazené posloupnosti barev pro vzorový příklad**

Aby program nemusel procházet vždy všechny posloupnosti při hledání, zda už se na nějaké kostce daná posloupnost vyskytla, vytvoříme pole a jednoduchou hashovací funkci. Jejím vstupem bude součet čtyř barev a výstupem tento součet vydělený modulo velikostí pole, v němž jsou seznamy se stejným hashem. Kombinace s největším počtem pak určuje čtyři stěny výsledné věže (viz Obrázek 7).



**Obrázek 7 - Postavená věž ze vzorového příkladu**

## 3.7 Superpočítač (1153)

### Zadání

Na Katedře informatiky a výpočetní techniky na Západočeské univerzitě sestavili nový superpočítač. Při testování rychlosti má sečíst prvních $N$ ($N < 10^{600}$) přirozených čísel. Bohužel poté, co počítač dospěl k výsledku, student zadávající číslo $N$ zapomněl, které to bylo.

Pomozte nebohému studentovi přijít na to, o jaké číslo se jedná.

### Vstup

Součet prvních $N$ přirozených čísel.

### Výstup

Číslo $N$, které bylo zadáno.

### Příklad

vstup:

28

výstup:

7

## Řešení

Při řešení tohoto příkladu využijeme faktu, že celá část odmocniny z dvojnásobku součtu $N$ přirozených čísel je rovna $N$. Dvojnásobek součtu ovšem nemůžeme získat pouhým vynásobením, jelikož neexistuje datový typ, který by takto velké číslo obsáhl.

Načteme tedy vstupní číslo jako řetězec a rozdělíme jej (zprava) do pole po dvojicích číslic (jako poslední může zbýt jen jedna číslice). Poté všechna čísla v poli vynásobíme dvěma a posouváme se od posledního indexu pole k prvnímu, přičemž pokud vzniklo v některé buňce pole číslo trojciferné, necháme v této buňce pouze číslo menší než 100 a vyšší hodnoty převedeme do další buňky pole.

Nyní potřebujeme odmocnit číslo zapsané po dvojicích číslic v poli. V cyklu procházíme pole od nejnižšího indexu a používáme následující algoritmus [14]:

1. Pro první číslo $x_1$ z pole najdeme takové číslo $0 < y < 10$, kdy platí $y^2 \leq x_1 < (y+1)^2$. Číslo y je první číslicí výsledku.
2. Od čísla $x_1$ odečteme $y^2$ a rozdíl vynásobíme 100, následně k němu přičteme další číslo z pole. Výsledné číslo označíme jako $x$.
3. Nyní označme prozatímní výsledek jako $v$. Hledáme číslo $?$, které by vyhovovalo vztahu:

$$(1) \qquad (20v+?)\cdot ? \leq x < (20v+(?+1))\cdot(?+1)$$

Nalezené číslo je další číslicí výsledku. Dalším číslem x je pak výsledek rovnice

$$(2) \qquad x = (x-(20v+?)\cdot ?)\cdot 100 + \textit{další číslo z pole}$$

Algoritmus pokračuje v cyklu dokud se nespočtou všechny číslice výsledku.

Podívejme se nyní na konkrétní příklad a to pro vstup 861. Číslo rozdělíme do pole a poté vynásobíme dvěma (viz Tabulka 4).

| | | |
|---|---|---|
| Rozdělení čísla po dvojicích číslic | 8 | 61 |
| Vynásobení dvěma | 16 | 122 |
| Posun řádu | 17 | 22 |

**Tabulka 4 - Určení dvojnásobku čísla**

Nyní víme, že druhá odmocnina z 1722 je námi hledaný výsledek. Pro takto malé číslo by samozřejmě stačilo jednoduše odmocnit metodou programovacího jazyka, nicméně pro názornost ukažme použití algoritmu.

1. pro číslo 17 je hledaným číslem $y = 4$, protože platí $4^2 \leq 17 < 5^2$, první číslice výsledku je tedy 4.
2. počítáme číslo $x = (17 - 16) \cdot 100 + 22 = 122$
3. nyní hledáme číslo *?*, pro které platí vztah (1)
   $(20 \cdot 4 + ?) \cdot ? \leq 122 < (20 \cdot 4 + (?+1)) \cdot (?+1)$
4. hledané číslo je 1, jedná se o druhou číslici výsledku. Algoritmus ukončíme, jelikož v poli již žádné nepoužité číslo není. Výsledek je tedy $v = \sqrt{1722} = 41$

## 3.8 Obdélníky (1202)

### Zadání

Na nekonečně velkém listu papíru je systém souřadnic (jednotkou je 1 krok). Je na něm nakresleno $N$ obdélníků, jejichž hrany jsou rovnoběžné s osami $x$ a $y$ a jejichž hranice se mohou dotýkat. Pokud označíme souřadnice levého dolního rohu $i$-tého obdélníku jako $(x_i, y_i)$ a souřadnice pravého horního rohu jako $(x^i, y^i)$, zjistíme, že:

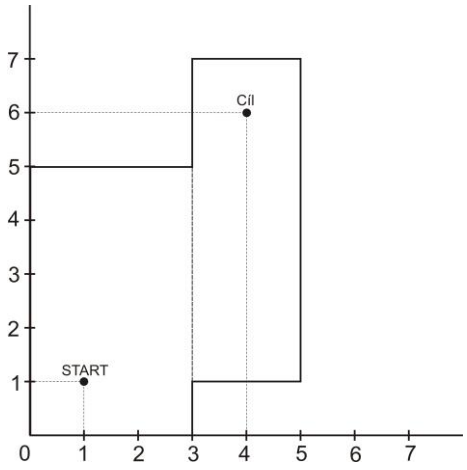
$x_1 = 0, y_1 = 0$

$x^i = x_{i+1}$

$2 \leq x^i - x_i \leq 100$

$2 \leq y^i - y_i \leq 100$

Pokud se hranice obdélníků $i$ a $i$+1 dotýkají, navzájem se protnou a zmizí (viz Obrázek 8).




**Obrázek 8 - Spojení dvou obdélníků**

Pomyslný cestovatel začal svou cestu na souřadnicích (1, 1), což je jistě uvnitř prvního obdélníku. Cestovatel se nesmí nikdy dostat až na hranici obdélníku, proto může dojít do jiného jen přes jejich zmizelou hranici. Cíl tohoto poutníka je na souřadnicích ($x^N - 1$, $y^N - 1$), což je jistě uvnitř posledního obdélníku.

## Vstup

Na první řádce je celé číslo $N$ ($0 < N < 100000$), počet obdélníků na papíru. Na následujících n řádcích je pak popis každého z obdélníků pomocí čtyř čísel: $x_i$, $y_i$, $x^i$, $y^i$ oddělených mezerami.

## Výstup

Počet kroků cestovatele nutných k cestě z bodu (1, 1) do bodu ($x^N - 1$, $y^N - 1$). Cestovatel jde nejkratší cestou (pokud nějaká existuje). Pokud cesta neexistuje, vypište -1.

## Příklad

vstup:

2

0 0 3 5

3 1 5 7

výstup:

8

### Řešení

Aby mohl cestovatel projít z jednoho obdélníku do dalšího, musí se jejich hrany dotýkat alespoň ve třech bodech souřadnic (projde tak uprostřed a nedotkne se žádné hrany). Otestujeme-li vždy po načtení nového obdélníku, zda je tato podmínka splněna, můžeme pokračovat dále, jinak se nelze dostat do cílového bodu a vypíšeme -1.

Pokud udržujeme celou dobu souřadnice cestovatele, na kterých se v danou chvíli nachází, může pro jeho přechod do dalšího obdélníku nastat jedna z následujících situací:

- posune se jen na souřadnicích $x$
- druhý obdélník má levý dolní roh výše, proto se musí cestovatel posunout i nahoru po ose $y$
- druhý obdélník má levý horní roh níže, proto se musí cestovatel posunout i dolů po ose $y$

Po vkročení do posledního obdélníku pak stačí, aby se cestovatel dostal do cílové pozice $(x^N - 1, y^N - 1)$.

## 3.9 Špión (1322)

### Zadání

Tajná služba objevila ve svém kruhu cizího agenta - špióna. Každý týden posílal nečitelné zprávy přímo přes internet. Aby bylo možné zjistit, jaký text zprávy obsahovaly, je nutné je dešifrovat. V agentově pokoji byl objeven stroj na šifrování, jenž pracuje podle následujícího postupu:

Jako vstup přichází do stroje jeden řetězec $S_1 = s_1s_2...s_N$. Stroj pak vytváří všechny cyklické permutace - $S_2 = s_2s_3...s_Ns_1$, ..., $S_N = s_Ns_1s_2...s_{N-1}$. Poté jsou všechny řetězce $S_1$, $S_2$, ..., $S_N$ vzestupně seřazeny podle abecedy a vypsány pod sebe každý na novou řádku. Tak vznikne dvourozměrné pole o rozměrech $N$ x $N$. Jedna z řádek tohoto pole obsahuje i vstupní zprávu. Číslo této řádky a poslední sloupec vzniklého pole jsou výstupem stroje.

Číslo ani délka řetězce nejsou větší než 10000. Řetězec může obsahovat jen znaky A-Z, a-z a podtržítko ('_'), lexikografické seřazení je následujícím způsobem:

ABCDEFGHIJKLMNOPQRSTUVWXYZ_abcdefghijklmnopqrstuvwxyz.

Například pro slovo $S_1$ = abracadabra vzniknou následující permutace:

1. aabracadabr = $S_{11}$
2. abraabracad = $S_8$
3. abracadabra = $S_1$
4. acadabraabr = $S_4$
5. adabraabrac = $S_6$
6. braabracada = $S_9$
7. bracadabraa = $S_2$
8. cadabraabra = $S_5$
9. dabraabraca = $S_7$
10. raabracadab = $S_{10}$
11. racadabraab = $S_3$

Výstupem pak bude číslo 3 a řetězec rdarcaaaabb.
Vaším úkolem je najít dešifrovací algoritmus těchto zpráv.

## Vstup

Na dvou řádcích je nejprve uvedeno výstupní číslo ze stroje a řetězec.

## Výstup

Zpráva, která se šifrovala.

## Příklad

vstup:
3
rdarcaaaabb

výstup:
abracadabra

## Řešení

Zmíněný postup kódování se nazývá Burrows-Wheelerova transformace [15]. Pro opětovné získání vstupního řetězce využijeme toho, že známe poslední sloupec. Z něho

lze snadno klasickým seřazením znaků vytvořit první sloupec. Rovnou tedy známe první a poslední písmeno výsledného řetězce, protože na vstupu je zadáno číslo správné řádky. Pro nalezení ostatních písmen si postup ukážeme rovnou na ukázkovém příkladu, jehož zadání je výše.

Mějme v Tabulce 5 dva známé sloupce, první a poslední:

| Index | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Poslední sloupec | R | d | a | r | c | a | a | a | a | b | b |
| První sloupec | A | a | a | a | a | b | b | c | d | r | r |

**Tabulka 5 - Sloupce pro řetězec abracadabra**

Znaky obou sloupců se stejným indexem budou ve výstupním řetězci v tomto pořadí (víme tedy, že výsledný řetězec, nebo alespoň jeho cyklický tvar, dvakrát obsahuje “ra“, dvakrát “ab“ atd.). Bohužel zatím nejsme schopni určit, v jakém pořadí za sebou tyto dvojice poskládat.

Odbočíme-li teď trochu od algoritmu a podíváme se na všechny cyklické permutace řetězce abracadabra v zadání, zjistíme, že pro první znak ‘a‘ prvního sloupce je řetězcem “aabracadabr“, pro první znak ‘a‘ v posledním sloupci je řetězec také “aabracadabr“ (po cyklickém posunu, aby byl tento znak umístěn jako první v řetězci). Totéž platí pro druhé ‘a‘, ale i třetí a obecně pro všechny znaky. Pokud je tedy více znaků stejných, platí, že jsou pro poslední i první sloupec zaneseny do řetězců ve stejném pořadí.

Díky tomuto pravidlu můžeme zjistit předchůdce pro konkrétní znaky. Víme například, že pro první znak ‘a‘ v posledním sloupci (index 3) je předchůdce stejný jako pro první znak ‘a‘ ve sloupci prvním (index 1). Do Tabulky 6 zaneseme tyto hodnoty pro zpětnou transformaci. V tabulce je naznačeno, jakým způsobem jsme přišli na první dvě hodnoty.

| Index | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Poslední sloupec | R | d | a | r | c | a | a | a | a | b | b |
| První sloupec | A | a | a | a | a | b | b | c | d | r | r |
| Předchůdce | 10 | 9 | 1 | 11 | 8 | 2 | 3 | 4 | 5 | 6 | 7 |

**Tabulka 6 - Předchůdci znaků**

Pokud začneme provádět transformaci od prvního znaku posledního sloupce, získáme řetězec “rbadacarbaa“. Tento řetězec musíme ještě zrcadlově otočit, protože další znaky při transformaci nejsou následovníci prvního, ale jeho předchůdci. Máme tedy řetězec “aabracadabr“. Vidíme, že se jedná o jednu z cyklických permutací výsledku. Správný řetězec získáme, pokud budeme transformaci provádět rovnou od posledního znaku, o němž díky vstupní hodnotě řádky víme, že se nachází na indexu 3.

## 3.10 Balíček karet (1244)

### Zadání

Je dán neúplný balíček karet. Vaším úkolem je napsat program, jenž určí, které z karet v balíčku chybí.

### Vstup

První řádka obsahuje kladné celé číslo $M$ - váha neúplného balíčku v miligramech. Na druhé řádce je číslo $N$ ($2 \leq N \leq 100$) - počet karet v kompletním balíčku. Následuje $N$ řádek, kdy v každé z nich je hmotnost dané karty v rozmezí 1 až 1000.

### Výstup

Pokud:

- neexistuje řešení, vypište 0.
- existuje více než jedno řešení, vypište -1.
- existuje právě jedno řešení, vypište čísla karet, která v balíčku chybí ve vzestupném pořadí a oddělená mezerami.

### Příklad

vstup:

14

4

10

7

4

6

výstup:

2 4

## Řešení

Po načtení vstupních údajů si vytvoříme dvourozměrnou matici o velikosti (*M*+1) x 2, kde *M* je hmotnost neúplného balíčku. Počáteční hodnoty matice budou 0 až na pozici (0,0), kterou nastavíme na -1. Postupně pak pro každou kartu procházíme celou matici od konce k začátku, a pokud narazíme na nenulovou hodnotu buňky, přiřadíme index dané karty do buňky s indexem o hmotnost karty vyšším. Pro lepší názornost viz následující řešení příkladu v Tabulkách 7-11:

| Počet karet | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Index karty | -1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Hmotnost | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |

**Tabulka 7 - Počáteční inicializace matice**

| Počet karet | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Index karty | -1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| Hmotnost | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |

**Tabulka 8 - Vložena první karta**

| Počet karet | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Index karty | -1 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| Hmotnost | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |

**Tabulka 9 - Vložena druhá karta**

| Počet karet | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Index karty | -1 | 0 | 0 | 0 | 3 | 0 | 0 | 2 | 0 | 0 | 1 | 3 | 0 | 0 | 3 |
| Hmotnost | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |

**Tabulka 10 - Vložena třetí karta**

| Počet karet | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 2 | 1 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Index karty | -1 | 0 | 0 | 0 | 3 | 0 | 4 | 2 | 0 | 0 | 1 | 3 | 0 | 4 | 3 |
| Hmotnost | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |

**Tabulka 11 - Vložena čtvrtá karta**

V Tabulce 11 vidíme, že pro hmotnost 14 (tj. neúplný balíček karet) je počet karet 1. Z toho lze usoudit, že určitě existuje řešení. Pokud by zde byla hodnota 0, řešení by neexistovalo, zrovna tak pokud by byla hodnota vyšší než 1, existovalo by jistě více variant pro seskládání tohoto balíčku. Takhle ale nemůžeme s jistotou říci, zda je více kombinací či nikoli.

Nyní postupně projdeme matici od konce. Víme, že se balíček jistě skládá z karty s indexem 3, ta váží 4 mg. Po odečtení 14 – 4 se dostáváme k dalšímu indexu karty na pozici 10, tentokrát je to číslo 1. Karta číslo 1 váží 10 mg, po odečtení jsme tedy už na hmotnosti 0. Možnou variantou karet, z kterých se skládá neúplný balíček jsou čísla 1 a 3.

Nyní provedeme dvakrát znovu algoritmus použitý na začátku příkladu s tím rozdílem, že jednou použijeme karty objevující se v balíčku a podruhé ty ostatní.

| Počet karet | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Index karty | -1 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 2 |
| Hmotnost | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |

**Tabulka 12 - Karty z balíčku**

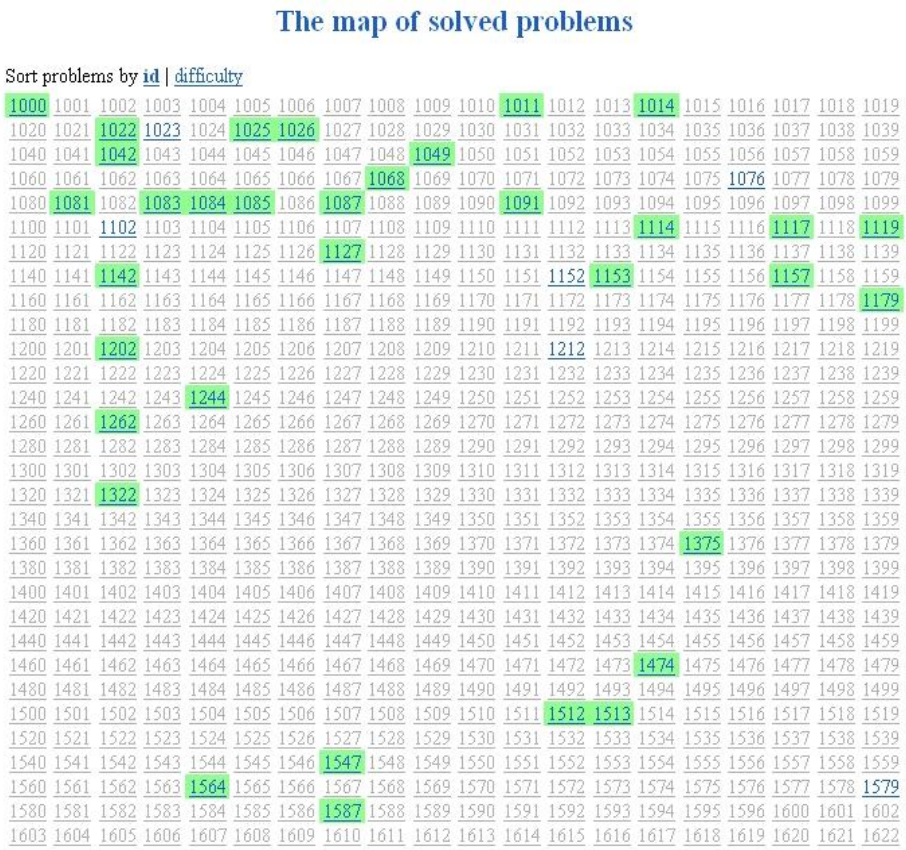
| Počet karet | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Index karty | -1 | 0 | 0 | 0 | 0 | 0 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 2 | 0 |
| Hmotnost | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |

**Tabulka 13 - Karty, které nejsou v balíčku**

Abychom mohli potvrdit, že je výstup jednoznačný, nesmí žádná kombinace karet, které nejsou v balíčku, nabýt hodnoty jako jakákoli kombinace karet, jež v balíčku jsou. Z tabulek 12, 13 je zřejmé, že v tomto případě je výsledek jednoznačný, karty s čísly 2, 4 nejsou v balíčku.

## 3.11 Souhrn

Na serveru Timus jsem dohromady vyřešil třicet úkolů (na Obrázeku 9 označeny zeleně), které jsem přepracoval do podoby použitelné v soutěži PilsProg. Nesnažil jsem se o co možná největší počet úloh, spíše jsem se zaměřil na jejich různorodost. Všechny úlohy včetně postupů řešení a testovacích souborů jsou na přiloženém CD. K některým úlohám jsem také kromě vstupních souborů vytvořil program na jejich generování.




**Obrázek 9 - Část mapy úloh na serveru Timus**

# 4 ZÁVĚR

Při práci jsem se nejprve důkladně seznámil s mezinárodní soutěží pro studenty ACM-ICPC pořádanou společností Association for Computer Machinery. Jejímu principu je velmi podobná česká soutěž PilsProg, kterou organizuje Katedra informatiky a výpočetní techniky Fakulty aplikovaných věd na Západočeské univerzitě v Plzni. Hlavním rozdílem je kladení důrazu na individuální schopnosti u soutěže PilsProg narozdíl od týmové spolupráce v ACM-ICPC. Pro trénování na obě soutěže mohou studenti využít různých validačních serverů jako je Timus Online Judge či Sphere Online Judge, kde je množství úloh k řešení.

Server Timus jsem dále důkladněji popsal. Úlohy jsou zde rozděleny do různých kategorií (řešitelné pomocí grafů, dynamického programování atd.) a dají se seřadit podle obtížnosti. Jednotlivá zadání lze řešit v programovacích jazycích Java, C, C++, C# a Pascal a po kontrole server vrátí odpověď, zda byl program správným řešením, na některý vstup odpověděl chybně či se ho vůbec nepodařilo přeložit.

V další části práce jsem si na serveru Timus vybral třicet úloh, které jsem vyřešil a přepracoval, aby se daly použít v soutěži PilsProg. Úlohy jsem volil takové, aby pokryly různé stupně obtížnosti a daly se tak použít nejen v kvalifikačních kolech soutěže, ale i ve finále. Ke každému zadání jsem vytvořil český překlad a sadu testovacích vstupů a výstupů určujících správnost kontrolovaných programů.

Po těžkopádných začátcích při řešení některých úloh jsem se při práci naučil zejména předem si rozmyslet, jaké jsou všechny možné cesty vedoucí k cíli. Vybírání toho nejlepšího způsobu a postupné zdokonalování jednotlivých kroků algoritmu mi pomohlo k (později) mnohem přesnějšímu odhadu náročnosti úkolů ještě před tím, než jsem se do něho vůbec pustil.

Bohužel ani já ještě nejsem natolik zkušeným programátorem, abych nyní zvládl i ty nejtěžší úlohy skrývající se na serveru Timus. Postupným vylepšováním základních dovedností a nalézáním pro mě dosud neznámých algoritmů ale věřím, že se mé schopnosti budou dále zlepšovat.

# Seznam zkratek

1. ACM - Association for Computer Machinery
   Mezinárodní společnost zabývající se výpočetní technikou.

2. ACM-ICPC – ACM- International Collegiate Programming Contest
   Programovací soutěž pořádaná společností ACM.

3. PilsProg
   Programovací soutěž v rámci České republiky pořádaná Katedrou informační a výpočetní techniky na Západočeské univerzitě v Plzni.

4. KIV – Katedra informatiky a výpočetní techniky
   Katedra na Fakultě aplikovaných věd

5. UVA – The Universidad de Valladolid Robot Judge
   Jeden z ACM validačních serverů.

6. SPOJ - Sphere Online Judge
   Jeden z ACM validačních serverů.

7. Timus – Timus Online Judge
   Ruský ACM validační server.

# Literatura, elektronické odkazy a zdroje


[1] SEDGEWICK, Robert. *Algorithms in Java. Pts. 1-4, Fundamentals, data structures, sorting, searching*. 3rd ed. Boston: Addison-Wesley, c2003. xix, 737 s. ISBN 0-201-36120-5.

[2] WRÓBLEWSKI, Piotr. *Algoritmy: datové struktury a programovací techniky*. Vyd. 1. Brno: Computer Press, 2004. 351 s. ISBN 80-251-0343-9.

[3] SKIENA, Steven S. a REVILLA, Miguel A. *Programming challenges: the programming contest training manual*. New York: Springer, c2003. xix, 359 s. Texts in computer science. ISBN 0-387-00163-8.

[4] HEROUT, Pavel. *Učebnice jazyka Java*. 3., rozš. vyd. České Budějovice: Kopp, 2007. 381 s. ISBN 978-80-7232-323-4.

[5] ACM
URL: <http://www.acm.org/>
[citováno: 2012-4-1]

[6] ACM-ICPC
URL: <http://cm.baylor.edu/welcome.icpc>
[citováno: 2012-4-1]

[7] Central European Regional Contest
URL: <http://contest.felk.cvut.cz/11cerc/>
[citováno: 2012-4-1]

[8] PilsProg
URL: <http://pilsprog.fav.zcu.cz>
[citováno: 2012-4-1]

[9] Seznam validačních serverů
URL: <http://www.algorithmist.com/index.php/List_of_Problemsets>
[citováno: 2012-4-1]

[10] The Universidad de Valladolid Robot Judge
URL: <http://uva.onlinejudge.org/>
[citováno: 2012-4-1]

[11] Sphere Online Judge
URL: <http://www.spoj.pl>
[citováno: 2012-4-1]

[12] Timus Online Judge
URL: <http://acm.timus.ru/>
[citováno: 2012-4-1]

[13] Fibonacciho posloupnost
URL:< http://mathworld.wolfram.com/FibonacciNumber.html>
[citováno: 2012-4-13]

[14] Algoritmus pro odmocnění
URL:<http://www.homeschoolmath.net/teaching/square-root-algorithm-example.php>
[citováno: 2012-4-1]

[15] Burrows-Wheelerova transformace
URL: <http://alivebutsleepy.srnet.cz/burrows-wheelerova-transformace/>
[citováno: 2012-4-1]