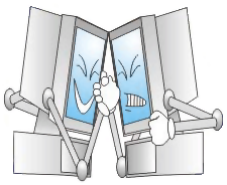


Окружно такмичење из информатике

Анализа проблема са решењима

Андреја Илић
e-mail: andrejko.ilic@gmail.com

Никола Милосављевић
e-mail: nikola5000@gmail.com

Природно математички факултет, Ниш

март 2012. година

Окружно такмичење из информатике, у циклусу школске 2011/2012. године, одржано је 03. марта. Укупно је учествовало 380 такмичара (отприлике трећина из А и две трећине из Б категорије), што показује значајно веће интересовање за овај тип такмичења у односу на претходне године. Структура такмичења је остала иста као и прошлих година: такмичар је имао четири сата за три алгоритамска проблема.

Нажалост, и поред квалификација и пропратне приче, велики број такмичара је направио "руки" грешке. Овде спадају грешке везане за погрешне називе изворних фајлова или датотека, комуникација преко стандардног улаза односно излаза, остављање `system('pause')` и `readln` на крају кода, ограничења низова, типови података... Ово су, сложићемо се, јако наивне грешке али недопустиве. Многи такмичари су, нажалост, због "само" пар бајтова кода изгубили доста бодова (неки чак и цео проблем). И овом приликом, још једном (по ко зна који пут), апелујемо на све такмичаре да на овакве ситне грешке обрате посебну пажњу и да ово науче на туђим а не на својим превидима.

Задаци су се показали како доста тешки (иако се очекивало супротно). Просечан број бодова по задацима је приказан на графику испод. Броја такмичара са максималним учинком, по задацима, је 39, 15, 59, 10 и 2, редом.

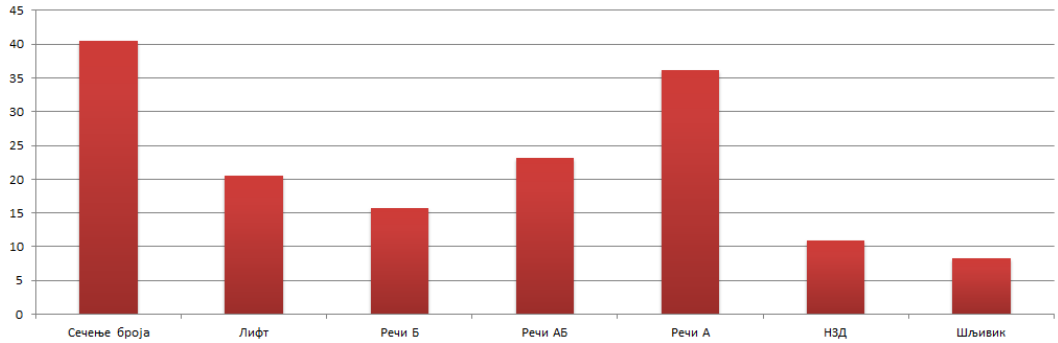


График 1. Просечан број бодова по задацима.

Просечан број освојених бодова у Б категорији се повећао у односу на прошлу годину. Но, овде је очекиван много бољи резултат у проблему Сечење броја (јер је више "школски" тип проблема). Са друге стране, резултати А категорије су јако лоши. Проблеми НЗД и Шљивик су се показали као доста тврд орах за такмичаре. Доста такмичара је поклекла у имплементацији идеја - ово је изгледа велики проблем наших такмичара (изгледа да имплементацију сматрају "рутинским" делом процеса решавања али резултати показују нешто друго).

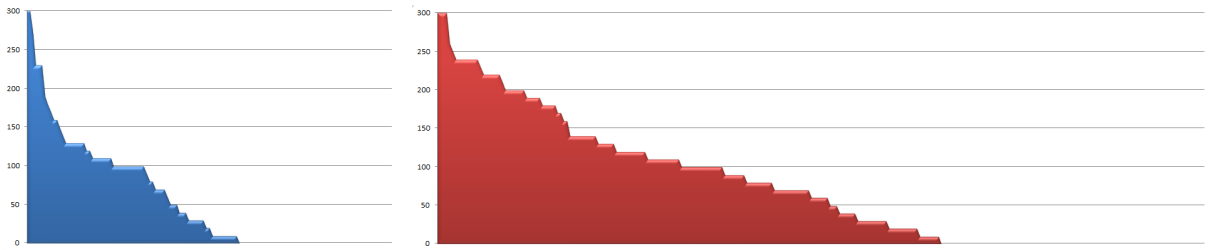


График 2. График броја бодова по категоријама.

При тестирању решења такмичара, као и у току израде овог материјала, аутори су наишли на доста занимљивих ствари у кодовима. **Додатак: Занимљивости из кодова такмичара**, који је дат на крају овог билтена, саджи неке од ових њих.

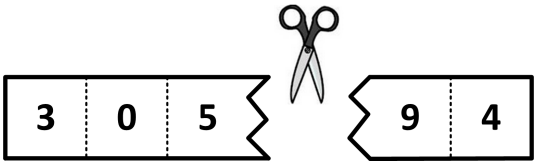
Аутори ову верзију билтена сматрају рандом верзијом. Такмичаре, и све који се тако осећају, би овом приликом замолили да уколико уоче неке грешке (којих сигурно има) или имају додатне идеје / коментаре на решења и проблеме, да нам се обрате путем маила.

Није знање, знање знати, већ је знање, знање другом дати.

Проблем 1. Сечење броја

Дата су два природна броја n и m. Број n можемо записати у декадном запису преко низа цифара као $n = \overline{c_k c_{k-1} \dots c_2 c_1}$.

Над бројем n је могуће применити операцију сечења између неке две узастопне цифре. Ако број n исечемо на два дела између цифара $i+1$ и i, $i \in [1, k-1]$ добијамо два нова броја $n_1 = \overline{c_k \dots c_{i+1}}$ и $n_2 = \overline{c_i \dots c_1}$. При овом сечењу дозвољен је и случај када n_2 почиње водећим нулама, тј. када је $c_i = 0$.



Пример сечења броја $n = 30594$ при чему се добија $n_1 = 305$ и $n_2 = 94$.

За дате бројеве n и m наћи сечење броја n такво да је апсолутна разлика суме добијених делова, $n_1 + n_2$, и броја m минимална. Другим речима, наћи сечење које минимизира израз

$$|m - (n_1 + n_2)|$$

Улаз. (Улазни подаци се налазе у датотеци `sek-sek.in`) У првом и једином реду улазне датотеке налазе се два природна броја n и m ($10 \leq n \leq 10^{18}$, $1 \leq m \leq 10^{18}$) описана у тексту проблема. Бројеви n и m немају водећих нула.

Излаз. (Излазне податке уписати у датотеку `sek-sek.out`) У првом и једином реду излазне датотеке исписати тражену вредност (минималну апсолутну разлику броја m и суме делова неког сечења броја n).

Пример 1.

`sek-sek.in`	`sek-sek.out`
`30594 400`	`1`

Објашњење. Сва могућа сечења и одговарајуће апсолутне разлике за дати пример су:

$n_1 = 3$ и $n_2 = 0594$	197
$n_1 = 30$ и $n_2 = 594$	224
$n_1 = 305$ и $n_2 = 94$	1
$n_1 = 3059$ и $n_2 = 4$	2663

Решење и анализа. Означимо са k број цифара броја n. Ограничења у задатку су прилично мала - за k имамо да важи $k \leq 18$. Зато можемо за сваку од могућих $k-1$ позиција извршити сечење и израчунати добијену апсолутну разлику. Уколико је она мања од тренутно најбољег решања, тренутно решење постављамо на њену вредност.

Нека смо при сечењу између цифара i и $i+1$ добили делове n_1 и n_2. Како на основу ових вредности можемо "брзо" добити делове за наредно сечење, између цифара $i+1$ и $i+2$? Уколико цифре број n означимо са c_i као и у тексту проблема, имамо да је $n_1 = \overline{c_k c_{k-1} \dots c_{i+2} c_{i+1}}$. Наредну вредност за n_1 добијамо простим целобројним дељењем са 10, јер, уствари, само треба избацити последњу цифру c_{i+1}. Са друге стране, како је $n_2 = \overline{c_i \dots c_2 c_1}$, овде треба додати управо избачену цифру c_{i+1} из број n_1. Њу додајемо као цифру највеће тежине, јер нова вредност број n_2 треба бити $\overline{c_{i+1} c_i \dots c_1}$. Ово рачунамо као: $10^i \cdot c_{i+1} + n_2$. Дакле, у сваком кораку треба да одржавамо степен броја 10. За почетне вредности промењивих n_1 и n_2 узимамо n и 0, редом.

Алгоритам: Псеудо код проблема Сечење броја

```
Input: природни бројеви n и m
Output: тражена минимална апсолутна разлика

toReturn = ∞;
n1 = n;
n2 = 0;
degree = 1;
while (n1 ≠ 0) do
    c = n1 MOD 10;
    n2 = n2 + degree · c;
    n1 = n1 DIV 10;
    degree = 10 · degree;
    tmp = |m − (n1 + n2)|;
    if (tmp < toReturn) then
        toReturn = tmp;
    end
end
return toReturn
```

У горе описаном псеудо коду, резултат *toReturn* на почетку постављамо на неку велику вредност. Под овим подразумевамо вредност која је сигурно већа од решења, тј. у нашем случају можемо узети да је $\infty = 10^{18}$.

Овде треба бити пажљив са избором типа промењивих. Наиме, ограничење дато у тексту проблема захтева 18 значајних цифара. Зато треба промењиве дефинисати као *long long* у *C*-у, односно *QWord* у *Pascal*-у. Међутим, такмичари који раде у *C++*-у треба да обрате пажњу на *abs* функцију. Наиме, не постоји верзија ове функције која као резултат враћа вредност *long long* типа. Због овога, она се мора имплементирати у самом коду.

Сложеност овог алгоритма је занемарљива. Уколико узмемо апроксимацију за број цифара природног број n, $k \approx \log n$, добијамо да је сложеност овог алгоритма $O(k) \approx O(logn)$.

Грешке такмичара. Иако ово представља стандардан проблем који се ради и на школским часовима, доста такмичара није успело да уради овај проблем. Главни проблем биле су грешке у имплементацији (многе настале због компликованог приступа). Овде износимо неколико најфреквентнијих грешака:

- **тип промењивих**: Ово је грешка коју смо, нажалост, очекивали (но потајно се надали да неће бити оволико заступљена). Многи такмичари су користили тип промењиве који не подржава ограничења за бројеве n и m. Било је и пар имплементација у којима су улазне промењиве имале одговарајући тип, али су међу резултати дефинисани погрешно.

Иако ово можда звучи већ и досадно увек али **увек али УВЕК проверити ограничења за улазне и излазне као и помоћне промењиве**. Грешке овог типа су, сложићемо се, глупе али кошатају много. Пре имплементације обратите пажњу на ово (и док читате сам проблем), јер скоро увек је потребно само променити тип промењиве или величину низа или матрица.

- **гранични случајеви**: Доста алгоритама је изоставило случај када је n_1 или n_2 само једна цифра. Разлог за ово је вероватно "покушај" да се увек обезбеди сечење броја. Ова грешка би се свакако избегла једноставним исписивањем свих вредности које узимају ове две промењиве при испитивању свих позиција за сечење у алгоритму.
- **дискретно сечење**: Означимо са num_n и num_m број цифара бројева n и m, редом. Претпоставимо да је $num_n > num_m$. Пар алгоритама је за овај случај враћало резултат добијен сечењем између цифара num_m и $num_m + 1$. Но, ово не мора увек бити тачно. Свакако желимо да збир бројева $n_1 + n_2$ буде "што ближег" реда величине као и број m. Поред овог треба посматрати и случај $num_n - num_m$ и $num_n - num_m - 1$ (због истог резона). Али чак ни ово није довољно, јер водеће нуле број n_2 могу доста утицати на описани ред величине. Такође, овај проблем није ни захтевао егзактно тражење решења тако да није било потребе за овим.

Тестирање. Такмичарска решења су тестирана на корпусу од 10 тест примера. За овај проблем није било потребе за неким "специјалним" случајевима. Можда треба напоменути да је било случајева када је решење 0, када бројеви n и m прелазе ограничење од 2 милијарде и када је број n двоцифрен.

РБ	Вредност n	Вредност m	Решење
01	123	20	4
02	10	7	6
03	10000	10969	9969
04	10100	101	0
05	1000000001	65235	34766
06	1523640541	4031165	390472
07	1234567890	1	80234
08	99999999999999999	9999999999999999	9
09	10230405000690081	43215601	32295115
10	94013500480301400	3500480	477741055

Табела 1. Вредности улаза и решења тест примера за проблем Сечење броја.

Проблем 2. Лифт

У Бајтограду се отвара нови тржни центар. На отварање је дошло n становника Бајтограда. Тржни центар је огроман и има више спратова. Сваки спрат има другачије продавнице. Људи су упознати које продавнице су на ком спрату, па је свако у складу са тим одлучио који спрат ће прво да обиђе.

Међутим, власници тржног центра нису очекивали толики број људи, па су поставили само један лифт. Срећом, тај лифт је из нове серије B2512 супер-брзих лифтова. Лифту је потребна само једна секунда да се попне или спусти за један спрат. У лифт може да стане највише c особа.

Лифт, као и сви људи, на почетку се налазе у приземљу - на спрату 0. Ако је познато на који спрат свака особа жели да оде и ако је време заустављања лифта занемарљиво, одредите колико је минимално време потребно да све особе оду на жељени спрат.

Улаз. (Улазни подаци се налазе у датотеци `lift.in`) У улазној датотеци се у првом реду налазе два броја n и c ($1 \leq n, c \leq 1.000$), број људи и капацитет лифта, респективно. У следећем реду налази се n бројева из интервала $[0, 100.000]$, који представљају на који спрат свака од n особа жели да оде.

Излаз. (Излазне податке уписати у датотеку `lift.out`) У првом и једином реду излазне датотеке исписати минимално време у секундама потребно да свака особа стигне на жељени спрат.

Пример 1.

lift.in	lift.out
4 2 1 2 4 2	8

Објашњење. У лифт уђу две особе које желе да иду на 2. спрат. Лифт потроши 4 секунде да се попне до 2. спрата и врати до приземља. Затим у лифт уђу особа која иде на 1. спрат и особа која иде на 4. спрат. Лифт се попне до 1. спрата на ком изађе једна особа и затим настави до 4. спрата где изађе и друга. За то је потребно још 4 секунде, па је укупно време 8 секунди.

Пример 2.

lift.in	lift.out
5 3 6 4 6 4 2	14

Објашњење. У лифт уђу три особе које иду на спратове 2, 4 и 4. Лифт потроши 4 секунде да остави особе на жељеним спратовима и 4 секунде да се врати на приземље. Затим у лифт уђу две особе које иду на 6. спрат, те лифт потроши још 6 секунди да дође до 6. спрата. Укупно време је 14 секунди.

Решење и анализа. Означимо са A_i - редни број спрата на који жели да оде i-та особа. Пробајмо прво да изведемо неколико **интуитивних** закључака и идеја везаних за задатак

(често је лакше скупљати чињенице успут, у деловима):

- Превоз једне групе људи изгледа тако што лифт покупља $k \leq c$ особа, развози их по спратовима и, уколико је преостало неразвезених људи, он се враћа назад.
- За превоз једне групе људи, лифту треба $2 \cdot A$ секунди, где је A редни број највећег спрата на који иде нека особа из те групе, **осим** у случају када превози последњу групу - тада нема потребе да се враћа назад и потребно му је свега A секунди.
- Из претходног, закључујемо да у оквиру групе није битно којим редоследом лифт истоварује особе. Такође, редослед група није битан, осим последње - пошто тада нема враћања, "паметно је" да лифт до највишег спрата оде у тој последњој групи.
- Веће групе људи $\Rightarrow$ мање превоза група људи $\Rightarrow$ мање сабирања времена. Према томе "интуитивно је јасно" да величина сваке групе треба бити што већа, тј. да све, осим евентуално једне, буду величине c.
- Уколико, од свих особа у некој групи, особа i иде на највиши спрат, остале се само "шлепају" и за њих се "штеди" време. Да би се постигла највећа уштеда, "јасно је" да те особе треба да буду са што виших спратова који су директно испод спрата i-те особе - тј. да се ради о узастопним спратовима.

На основу овога, имплементирамо следећи алгоритам: **сортирамо** низ A, а затим, почевши одоздо нагоре, узимамо по c особа из низа A и пребацујемо их лифтом, успут рачунајући времена. Наравно, ради провере сами смишљамо једноставан тест пример: $c = 2$ и $A = (2, 3, 1)$. По нашем алгоритму, поделили смо их у групе $\{1, 2\}\{3\}$ и укупно време је $2 \cdot 2 + 3 = 7$. Међутим, за поделу $\{1\}\{2, 3\}$ добијамо боље време $2 \cdot 1 + 3 = 4$. **У чему је грешка?!**

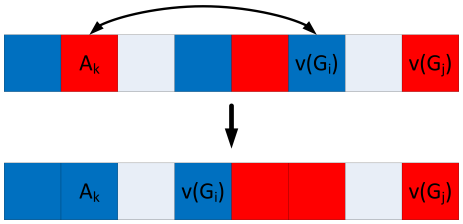
Није много чудно што је дошло до грешке; ништа од нашег *интуитивног* размишљања нисмо доказали (себи). Међутим, испоставља се да је 90% ове идеје добро - али нажалост, неретко (као и у овом случају) тих 10% који фали односи више од 50% поена. Кренимо од почетка, али нешто формалније.

Претпоставимо да је у оптималном решењу лифт превезао укупно m група $G_1, G_2 \ldots G_m$, тим редом. За сваку групу G назовимо *вођом групе* особу која иде на највиши спрат и означимо редни број тог спрата са $v(G)$. На основу ранијег запажања, тада лифт истроши укпуно $T = 2v(G_1) + 2v(G_2) + \ldots + 2v(G_{m-1}) + v(G_m)$ секунди. Пошто се времена свих група осим последње "дупло" рачунају, оптимално је да последња (m-та) група буде група G за коју је $v(G)$ највеће - у противном, заменимо последњу групу са оном која има већи $v(G)$ и добићемо боље време. Поредак осталих група није битан, па можемо претпоставити да је $v(G_1) \leq v(G_2) \leq \ldots \leq v(G_m)$ (овакве претпоставке треба користити јер упрошћавају сагледавање проблема).

Дакле, показали смо да **постоји** оптимално решење код кога за групе превоза важи $v(G_1) \leq \ldots \leq v(G_m)$. Шта можемо рећи о величини група? Докажимо претпостављено - да свака група, осим евентуално једне, увек садржи c особа. Претпоставимо да постоје групе G_i и G_j $(i < j)$, при чему G_j има мање од c особа. Тада пребацивањем вође групе G_i у групу G_j, вредност $v(G_i)$ се смањује или остаје иста а вредност $v(G_j)$ остаје иста (јер због $i < j$ је $v(G_i) \leq v(G_j)$. У сваком случају T не може да се повећа па ова пребацивања можемо да радимо докле можемо (налазећи нове парове група за које ово важи). Према томе, доћи ћемо до тренутка када не постоје индекси $i < j$ за које је $|G_j| < c$, тј. све групе $G_2, G_3, \ldots G_m$ ће имати тачно c људи.

Сада смо додатно доказали да **постоји** оптимално решење код кога је $v(G_1) \leq \ldots \leq v(G_m)$ и $|G_2| = |G_3| \ldots = |G_m| = c$. Приметимо да су тада m и $|G_1|$ јединствено одређени: $m = n \ div \ c$, $|G_1| = n \ mod \ c$, где су div и mod целобројно дељење и остатак, редом.

За крај, докажимо да *свака* од датих група може бити баш скуп од неколико узастопних спратова A_i из сортираног низа A, тј. да је $G_i = \{A_{x_i}, A_{x_i+1}, \dots A_{x_i+k_i}\}$ за свако i. Доказ је опет прилично интуитиван: нека су G_i и G_j произвољне групе за које је $i < j$ и које се "укрштају", тј. за најмањи $A_k \in G_j$ важи $A_k < v(G_i) \leq v(G_j)$. Ако пребацимо вођу групе G_i у групу G_j а особу која иде на A_k-ти спрат пребацимо из G_j у групу G_i, тада $v(G_j)$ остаје исто а $v(G_i)$ се не повећава (Слика 1). С обзиром да се најмањи елемент групе G_j на овај начин стално повећава, ово можемо радити само коначан број пута. Када извршимо сва пребацивања за сваке две групе, добијамо управо групе које се састоје од узастопних елемената сортираног низа A (проверити!) чије је укупно време T не горе од почетног.



Слика 1. Избегавање "укрштених" група.

Ово је крај. За $m = n \; div \; c$ и $x = n \; mod \; c$, управо смо доказали да **постоји** оптимално решење за које важи $G_1 = \{A_1, A_2, \dots A_x\}, G_2 = \{A_{x+1}, A_{x+2}, \dots, A_{x+c}\}, \dots, G_m = \{A_{n-c+1}, \dots A_n\}$, где је A **сортирани низ**. Према томе, алгоритам је сличан првом (интуитивном) осим што се крећемо одозго надоле. Ово је уједно била и поменута грешка првог алгоритма - нигде (чак ни интуитивно) нисмо показали да група која има мање од c људи мора бити последња, док смо овде показали да то може бити прва група.

Алгоритам: Псеудо код за проблем Лифт

Input: c и низ A дужине n
Output: најмање време потребно да лифт развезе све особе

Сортирати растуће низ A;
$T = A[n]$;
$i = n - c$;

while *($i > 0$)* **do**
  $T = T + 2 \cdot A[i]$;
  $i = i - c$;
end

return T

Стално смо наглашавали реч *постоји* из разлога што смо само доказали *постојање* решења овог облика, а не да је *свако* решење овог облика (ово друго није ни тачно, рецимо у другом примеру са папира, и подела $G_1 = \{2, 2\}$, $G_2 = \{1, 4\}$ даје оптимално решење а не само подела $G_1 = \{1, 2\}$, $G_2 = \{2, 4\}$). Међутим, нама је постојање сасвим довољно.

Због ограничења за број n, за сортирање низа A се могао користити било који квадратни сорт (*selection sort*, *bubblesort*, ...). У другом делу сам пролаз кроз спратове је линеаран, па је укупна сложеност алгоритма $O(n^2)$. Уколико се користи *quicksort* или *counting sort* сложеност је $O(n \log n)$ и $O(n + max_A)$, редом.

Најчешћи број не-нула поена на овом задатку био је 40 и 100 - 40 је број поена који доноси *интуитивни* (погрешан) алгоритам. Очигледна поука: иако се на такмичењу тражи само код а не доказ, потребно је бар себи доказати зашто је алгоритам тачан.

Проблем 3. Речи

Мали Перица је недавно кренуо у школу и већ је научио да чита. Пошто је Перица веома вредан и савестан ученик, од своје учитељице Марије затражио је неки задатак где би могао да провери своје знање. Учитељица Марија, срећна што има тако доброг ученика, одмах му је дала један проблем на којем ће Перица проверити колико добро је савладао градиво.

Перица је добио мали речник R, који садржи n речи. Поред тога, учитељица му је дала и једну листу L од m речи. Задатак му је да тестира сваку реч из листе L на следећи начин: За сваку реч из речника пронаћи ће сва њена појављивања у речи коју тестира и прецртаће свако појављивање. На крају, Перица мора да саопшти својој учитељици колико речи из листе има свако своје слово прецртано бар једанпут.

Помозите учитељици Марији тако што ћете направити програм који одређује колико речи из листе L ће бити потпуно прецртано, како би она лакше проверила да ли је Перица добро урадио домаћи задатак.

Улаз. (Улазни подаци се налазе у датотеци `reci.in`) У улазној датотеци се у првом реду налази број n ($1 \leq n \leq 100$), број речи у речнику R. У следећем реду налази се n речи из речника. Речи су раздвојене једним размаком и нису дуже од 10 слова. У трећем реду налази се број m ($1 \leq m \leq 100$), број речи у листи L. Наредних m редова садрже по једну реч из листе L. Свака реч из листе неће бити дужа од 100 слова. Сва слова су мала слова енглеског алфабета.

Излаз. (Излазне податке уписати у датотеку `reci.out`) У првом и једином реду излазне датотеке потребно је исписати број потпуно прецртаних речи.

Пример 1.

`reci.in`
```
6
paja ram papa ma lama lig
7
papaja
salama
papaje
ramalama
mara
mama
miligram
```

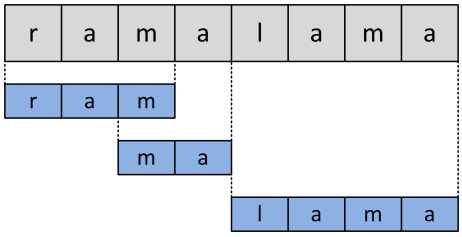
`reci.out`
```
3
```

Објашњење. Има 3 потпуно прецртане речи: папаја, рамалама, мама.

Решење и анализа. Проблем Речи је мање више симулација процедуре описане у тексту проблема. Као што видимо, свака реч из листе L се посматра засебно. Означимо тренутну реч из листа L са *word*. Алгоритам који ћемо сада изнети, примењујемо над сваком од речи из ове листе. Са $|w|$ означићемо дужину речи w, како из речника R тако и из листе L.

Дефинисаћемо помоћни низ *mark* који ће бити типа *boolean*. Овај низ ће симулирати описано прецртавање у тексту проблема. За тип узимамо логичку вредност јер нам је само битно да

ли је слово прецртано или не, а није битно колико пута је прецртано. На почетку ни једно слово није прецртано односно низ *mark* је иницијализован на *false*. Сада редом обилазимо речи из речника. Сваку реч $t \in R$ покушавамо да "поставимо" у *word* на свим позицијама. Другим речима, за сваку позицији i, $i \in [1, |word| - |t| + 1]$, испитујемо да ли се подреч $word_i$ $word_{i+1}$... $word_{i+|t|-1}$ поклапа са t. Уколико се поклапа, у низу *mark* елементе на позицијама $\{i, i+1, \ldots, i+|t|-1\}$ постављамо на *true* (јер смо их прецртали помоћу речи t из речника).



Слика 1. Прекривања речи *ramalama* из примера са папира.

Након обиласка свих речи из речника, потребно је испитати да ли су сва слова прецртана бар једном. Ово је еквивалентно испитивању да ли су сви елементи низа *mark* постављени на *true*. Уколико јесу, реч *word* је потпуно прецртана, иначе није.

Алгоритам: Псеудо код за проблем Речи

```
toReturn = 0;
for i ← 1 to m do
    word = L[i];
    постави низ mark на false;
    for j ← 1 to n do
        t = R[j];
        for k ← 1 to |word| − |t| + 1 do
            if word_k word_{k+1} ... word_{k+|t|−1} = t then
                постави mark[p] = true за p ∈ [k, k + |t| − 1];
            end
        end
    end
    if (сви елементи низа mark су true) then
        toReturn = toReturn + 1;
    end
end
return toReturn
```

Која је сложеност овог алгоритма? Сваку реч из листе L смо посматрали посебно. За конкретну реч *word* за сваку позицију, којих има $|word|$, испитујемо да ли можемо поставити неку реч t из речника. Испитивање се своди на испитивање једнакости карактера којих има $|t|$. Дакле, укупна сложеност алгоритма једнака је $O(m \cdot n \cdot 100 \cdot 10)$.

Напомена. Проблем се може урадити и у сложености $O(m \cdot n \cdot 100)$ уколико за упоређивање користимо алгоритам КМП (енг. *Knuth Morris Pratt*). Међутим, овде треба бити пажљив при маркирању јер уколико би се ово радило као у описаном алгоритму сложеност би остала иста. Примера ради уколико је $word = aaa \ldots aa$ и $t = aa \ldots aa$ тада сваку позицију у низу *mark* постављамо на *true* $|t|$ пута уместо по једном. Када испитујемо конкретну реч t, памтимо само сегменте које треба маркирати у низу *mark*. Тек након испитивања свих позиција врши

се маркирање али тако да се сваки елемент низа *mark* постави највише једном (сортирамо по левим крајевима и док се крећемо по низу памтимо најдаљи десни крај).

Грешке такмичара. Проблем Речи, због природе решења који је симулација описаног поступка у проблему, није има много грешака. Већина такмичара је имало 0 или 100 бодова на мењу. Најчешћа грешка била је везана за иницијализацију помоћног низа *mark* - постављање на *true* је код неких имплементација ишло до $k + |t|$.

Међутим, на пар места се подкрала једна интересантна грешка. Наиме, у *C*-у крај стринга, знак \0, се третира као карактер. Из овог разлога, при дефинисању речи из речника треба узети ограничење од 11 карактера. Ово се не пријављује као грешка, јер се елементи низа третирају као узастопне меморијске локације. Иако приступамо елементу са индексом већим од ограничења низа, ми заправо приступамо меморијској локацији која је толико удаљена од адресе низа, односно у нашем случају стринга. Примера ради, уколико дефинишемо низ s као *char* $s[100][10]$ и у улазу имамо (за речи из речника): *aaaaaaaaaa bb*, имали би да је $s[0] = aaaaaaaaaabb$ и $s[1] = bb$. Овај "додатак" од два слова b је заправо стринг $s[1]$, јер када приступамо стрингу $s[0]$ крај стринга је први карактер \0 а он се налази након *bb*.

На крају, заинтересованог такмичара остављамо са једним интересантним проблемом односно "триком". Шта није уреду са наредним кодом?

```
#include <stdio.h>
#define MAXN 100

int array[MAXN], i;

int main()
{
    for (i = 0; i <= 100; i++)
    {
        array[i] = 0;
    }

    return 0;
}
```

Проблем 4. НЗД

Дат је низ природних бројева a од n елемената. У једном потезу дозвољено је одабрати произвољна два суседна броја у низу и **заменити их њиховим збиром**. На пример, уколико смо одабрали бројеве a_i и a_{i+1}, низ $(a_1, a_2, \ldots, a_i, a_{i+1}, \ldots a_n)$ постаје $(a_1, a_2, \ldots, a_i + a_{i+1}, \ldots a_n)$. Ово затим можемо понављати на новодобијеним низовима (приметимо да се број елемената низа сваки пут смањује за 1).

Применити одређени број потеза над почетним низом, тако да на крају добијемо **тачно** k бројева чији је највећи заједнички делилац **највећи могућ**.

Улаз. (Улазни подаци се налазе у датотеци `nzd.in`) У првом реду улазне датотеке налазе се 2 природна броја n и k који представљају, редом, број елемената у почетном низу и број елемената који треба остати на крају ($1 \leq k < n \leq 10^5$). Следећи ред садржи n природних бројева a_i који представљају почетни низ ($a_i \leq 10^6$). Сума свих бројева није већа од 10^6.

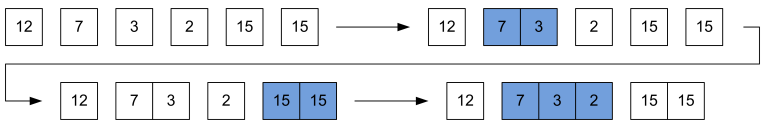
Излаз. (Излазне податке уписати у датотеку `nzd.out`) У првом реду излазне датотеке исписати максималан могућ највећи заједнички делилац преосталих бројева. У другом реду излазне датотеке исписати k природних бројева - изглед низа након свих потеза, чији је највећи заједнички делилац максималан. Уколико има више решења, штампати било које.

Пример 1.

nzd.in	nzd.out
6 3	6
12 7 3 2 15 15	12 12 30

Објашњење. Уколико извршимо потезе $(12, \mathbf{7}, \mathbf{3}, 2, 15, 15) \to (12, 10, 2, \mathbf{15}, \mathbf{15}) \to (12, \mathbf{10}, \mathbf{2}, 30) \to (12, 12, 30)$ добијамо бројеве 12, 12 и 30 чији је највећи заједнички делилац једнак 6. Није могуће добити 3 броја са већим највећим заједничким делиоцем.

Решење и анализа. Пробајмо да прво поједноставимо задатак. Рекурзивно понављање операције "замене два суседна елемента у низу њиховом сумом" наизглед делује незгодно за праћење и реконструкцију. Међутим, ово је прилично "стандардна" операција и не треба се концентрисати на редослед извршавања већ само на крајњи изглед низа. Замислимо да на почетку имамо n група, у i-тој групи број a_i. Ако поменуту операцију схватимо као спајање две суседне групе у једну (без брисања бројева) јасно је да се после сваке операције (па и на крају) свака група састоји од неколико **узастопних** елемената низа, као и да се сваки елемент низа налази у некој групи (види слику).



Слика 1. Креирање група на примеру са папира.

Како на крају свака група (тј. сума бројева у њој) представља један број, закључујемо да

смо, после примене одређеног броја операција, заправо поделили почетни низ на неколико узастопних поднизова и једноставно записали њихове суме у датом редоследу. Јако је битан и обрнут смер: било која подела низа на узастопне поднизове се може добити после примене одређеног броја ових операција (оставља се читаоцу за вежбу). Према томе, наш задатак је еквивалентан следећем: **нађи поделу датог низа на k узастпних поднизова тако да је највећи заједнички делилац свих вредности поднизова што већи**.

За дати природан број d, назовимо поделу низа на k узастопних поднизова, у којој d дели суму сваког подниза - *d-подела*. Суме поднизова ћемо у том случају означавати, редом, са $B_1, B_2, \dots B_k$. Решење задатка је тада највећи број d за који постоји d-подела. Заиста, ако је d највећи број за који постоји d-подела, тада је d уједно и $NZD(B_1, B_2, \dots B_k)$, иначе би NZD био већи број од d за који би постојала одговарајућа подела (користимо $d|NZD \Rightarrow d \leq NZD$). На овај начин смо елегантно "избацили" појам највећег заједничког делиоца из даљег тока решења.

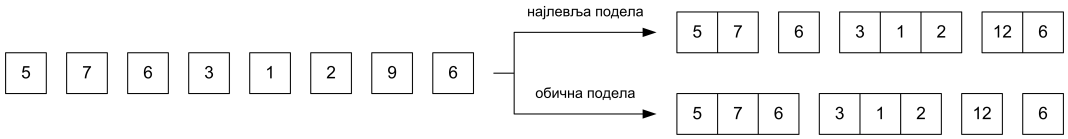
Често је, уместо тражења максималне вредности неког параметра (у нашем случају d) који мора да задовољава неке услове (да постоји d-подела), много лакше проверити да ли за дату конкретну вредност дати параметар задовољава услове (на томе се, рецимо, заснива бинарна претрага). Да ли можемо (и колико брзо) за **дату** вредност d проверити да ли постоји d-подела? Чак и да можемо, колико вредности за d треба проверити? Паметније је прво одговорити на друго питање, јер уколико је скуп могућих вредности за d превише велики, овај приступ неће дати резултата.

Претпоставимо да за неко d постоји d-подела. Нека је $S = a_1 + a_2 + \dots + a_n$. Приметимо да је, такође, $B_1 + B_2 + \dots B_k = S$. Како $d \mid B_i$ за $1 \leq i \leq k$ директно следи да је $d \leq S$. Ово одговара на последње питање - довољно је испитати S различитих вредности за d. Међутим, ово је јако груба оцена; из $d \mid B_i \Rightarrow B_i = d \cdot b_i$ па је $S = B_1 + B_2 + \dots B_k = (b_1 + b_2 + \dots + b_k) \cdot d$, тј. $d \mid S$. Према томе, уместо да проверавамо свако d из сегмента $[1, S]$, довољно је проверити **само делиоце броја** S - означимо њихов број са $\tau(S)$.

Из услова задатка $S \leq 10^6$ па је за очекивати да је $\tau(S)$ "не превише велики број". Заправо, сви делиоци броја S (као било ког другог природног броја) долазе у пару $(d, \frac{S}{d})$ где их можемо уредити тако да је $d \leq \frac{S}{d}$ па је $d \leq \sqrt{S}$. Следи да је $\tau(S) \leq 2\sqrt{S} \leq 2.000$. Међутим, у овој процени рачунамо да сваки број из $[1, \sqrt{S}]$ дели S што је превише грубо; испоставља се да је највећи број делиоца које има неки број $\leq 10^6$ једнак 240.

Вратимо се сада провери постојања d-поделе за *дати* број d. Наоружани сазнањем о величини скупа могућих вредности броја d, видимо да је алгоритам сложености $O(n)$ довољно добар за ову сврху и такав ћемо и дизајнирати.

Претпоставимо да за дато d постоји d-подела. Од свих таквих подела, уочимо *најлевљу* - поделу код које је B_1 најмање; у случају да има више подела са најмањим B_1 гледамо ону са најмањим B_2; затим B_3 итд. Јасно је да уколико постоји *нека* d-подела, онда постоји и најлевља (и обратно).



Слика 2. Пример најлевље и обичне поделе за случај $k = 4$ и $d = 6$.

Нека је i_1 најмањи индекс у низу a за који је $a_1 + a_2 + \dots + a_{i_1}$ дељиво са d. Докажимо да је тада $B_1 = a_1 + a_2 + \dots + a_{i_1}$. Заиста, уколико је $B_1 = a_1 + a_2 + \dots + a_{j_1}$, где је $i_1 < j_1$, тада због

$d \mid B_1$ и $d \mid (a_1 + a_2 + \ldots + a_{i_1})$, d дели и разлику ова два броја, тј. број $x = a_{i_1+1} + a_{i_1+2} + \ldots + a_{j_1}$. Међутим, тада је подела $(B_1 - x, B_2 + x, B_3, \ldots, B_k)$ такође d-подела и то левља од почетне, што је супротно претпоставци. Потпуно аналогно се доказује да је $B_2 = a_{i_1+1} + a_{i_1+2} + \ldots + a_{i_2}$, где је i_2 најмањи индекс за који је ова сума (која почиње од елемента a_{i_1+1}) дељива са d; такође и за $B_3, \ldots B_k$.

Ово нас наводи на следећи грамзиви (енг. *greedy*) алгоритам: поставимо вредност *currSum* на 0 и почињемо да се крећемо с лева на десно по низу a. Сваки пут када наиђемо на неки a_i, повећамо *currSum* за ту вредност. У тренутку када први пут $d \mid currSum$, повећавамо бројач група (нађених узастопних поднизова), ресетујемо *currSum* на 0 и настављамо даље. Уколико смо открили бар k група, налевља d-подела је нађена ("вишак" група на крају није проблем - све су дељиве са d и можемо их сажети у једну). Уколико смо дошли до краја са недовољним бројем група, тада, према разматрању у претходном пасусу, најлевља d-подела (а самим тим ни било која друга) не постоји.

Следећи псеудокод илуструје комплетан алгоритам. Водити рачуна да итерирамо по свим делиоцима суме почевши од **највећег**.

Алгоритам: Псеудо код за проблем Нзд

```
Input: n и низ a дужине n
Output: највеће d за који постоји d-подела, као и сама подела

S = a[1] + a[2] + ... a[n];
d = S + 1;
found = false;

while (not found) do
    d = d - 1;
    if (S MOD d = 0) then
        B[] ← 0;
        currSum = 0;
        groupCount = 0;

        for i ← 1 to n do
            currSum = currSum + a[i];
            if (currSum MOD d = 0) then
                groupCount = groupCount + 1;
                B[groupCount] = currSum;
                currSum = 0;
            end
        end

        if (groupCount ≥ k) then
            found = true;
        end
    end
end

for i ← k + 1 to groupCount do
    B[k] = B[k] + B[i];
end

return d, B[]
```

Приметимо да ће се **while** петља из псеудокода увек завршити јер је $d = 1$ вредност за коју сигурно постоји d-подела. Сложеност овог алгоритма је $O(S + \sqrt{S} \cdot n)$ - први део је због налажења свих простих бројева а други за проверу за свако $d \mid S$, где смо користили $\tau(S) \leq 2\sqrt{S}$. На основу поменуте чињенице о начину упаривања делиоца природног броја, за

налажење делиоца броја S можемо итерирати само до $\sqrt{S}$ и узимати d и $\frac{S}{d}$. Уз констатацију о броју делиоца бројева мањих од 10^6 добијамо сложеност од $O(\sqrt{S} + 240 \cdot n)$. Сви овакви алгоритми сложености $O(S + \tau(S) \cdot n)$ и бољи освајали су максималних 100 поена. Алгоритми сложености $(S \cdot n)$, који су испитивали свако d из сегмента $[1, S]$, освајали су 30 поена.

Још једна чињеница која може много убрзати алгоритам је следећа: из $S = B_1 + \ldots + B_k = (b_1 + \ldots + b_k) \cdot d \geq (1 + \ldots + 1) \cdot d = kd$ следи да је $d \leq \frac{S}{k}$ па је довољно итерирати по делиоцима броја S мањих од $\frac{S}{k}$. Ово је јако битно јер - ако је k велико тада радимо мање итерација; иначе имамо "више могућности" за груписање и добијају се већи бројеви d - опет мање итерација.

Проблем 5. Шљивик

Газда Срба има воћњак шљива у срцу Шумадије. Како је Срба велики перфекциониста, он свој шљивик одржава увек у форми правоугаоника, тако да у сваком од укупно N редова шљивика буде тачно по M стабала шљива. На сваком стаблу се налазе плодови, чији број газда Срба уредно контролише.

Срба је и поносни отац K деце, која обожавају шљиве, и да би их обрадовао, решио је да за свако дете набере исти број шљива. Међутим, пошто као и обично жели да све буде "под конац", газда Срба ће одабрати део воћњака који ће такође бити у облику правоугаоника, чије су странице паралелне страницама воћњака, али тако да, кад са тог дела набере све шљиве, он може све да их подели својој деци тако да свако од њих добије исти број шљива.

Одредити на колико начина газда Срба може да одабере део шљивика који ће да обере.

Улаз. (Улазни подаци се налазе у датотеци `sljivik.in`) У улазној датотеци се у првом реду налазе три броја N, M и K ($1 \leq N, M \leq 400$, $1 \leq K \leq 1.000.000$), број редова и колона Србиног воћњака и број Србине деце, респективно. У следећих N редова се налазе по M бројева из интервала $[1, 10^9]$ који представљају број плодова на сваком од стабала.

Излаз. (Излазне податке уписати у датотеку `sljivik.out`) У првом и једином реду излазне датотеке исписати број различитих начина да газда Срба одабере део шљивика који ће да обере.

Пример 1.

`sljivik.in`
```
3 3 5
2 9 3
10 8 6
1 4 12
```

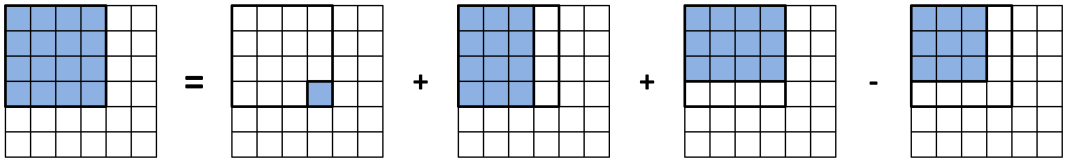
`sljivik.out`
```
4
```

Објашњење. Решења су ови правоугаоници (прва два броја су координате горњег левог угла, а друга два координате доњег десног угла правоугаоника са којег ће Срба да бере шљиве): (1,1)-(3,3) , (2,1)-(2,1) , (3,1)-(3,2) , (2,2)-(3,3).

Решење и анализа. Наивна имплементација, која за свака два елемента матрице (посматрамо их као горњи леви и доњи десни угао) сабира елементе ове подматрице, има сложеност $O(n^6)$. Наравно, ово је далеко од оптималног решења овог проблема.

Наивни приступ можемо убрзати дефинисањем помоћне матрице d. Елемент $d[i, j]$ дефинишемо као суму елемената подматрице чија су темена $(1, 1)$, $(i, 1)$, (i, j) и $(1, j)$. Како можемо брзо иницијализовати ову матрицу? Сума која се налази у елементу $d[i, j]$ се разликује од суме у елементу $d[i-1, j-1]$ за део i-те врсте и део j-те колоне. Примера ради, посматрајмо само део i-те врсте, тачније елементе $a[i, 1], \ldots, a[i, j-1]$. Суму за овај део врсте можемо добити као разлику: $d[i, j-1] - d[i-1, j-1]$. Аналогно, посматрани део за j-ту колону добијамо као: $d[i-1, j] - d[i-1, j-1]$. Коначно, добијамо да важи следећа рекурентна веза:

$$d[i, j] = a[i, j] + d[i-1, j-1] + (d[i, j-1] - d[i-1, j-1]) + (d[i-1, j] - d[i-1, j-1]) \quad (1)$$
$$= a[i, j] + d[i, j-1] + d[i-1, j] - d[i-1, j-1] \quad (2)$$



Слика 1. Начин рачунања елемента матрице d (плавом бојом су означени делови матрице чије суме посматрамо).

Из ове рекурентне везе, видимо да при рачунању елемента на позиции (i, j) користимо вредности са позиција $(i-1, j-1)$, $(i-1, j)$ и $(i, j-1)$. Ова поља нису дефинисана за вредности $i = 1$ или $j = 1$. Зато, прву врсту и прву колону матрице d треба посебно израчунати. Но, ово је једноставнији случај јер имамо само једну димензију. За дефинисање базе (део матрице који иницијализујемо на почетку како би могли да успоставимо рекурентну везу) користимо

$$d[1,1] = a[1,1]$$

$$d[1,j] = a[1,j] + d[1,j-1], j \in [2,n]$$
$$d[i,1] = a[i,1] + d[i-1,1], i \in [2,n]$$

Зашто нам је ово уопште било потребно? Зато што уз помоћ ове матрице d можемо у једној операцији рачунати суму елемената из подматрице (A_x, A_y) и (B_x, B_y). Сличним резоновањем као и за рекурентну релацију, добијамо да је ова сума једнака

$$sum(A_x, A_y, B_x, B_y) = d[B_x, B_y] - d[A_x - 1, B_y] - d[B_x, A_y - 1] + d[A_x - 1, B_y - 1]$$

Но овде опет имамо мали проблем: у случају када је $A_x = 1$ или $A_y = 1$ потребни су нам елементи из нулте колоне и врсте матрице d. Зато матрици d додајемо и нулту колону и врсту чији су елементи иницијализовани на нула.

Међутим, сада добијамо једну јако лепу ствар. Како за елементе $d[i, 0]$ и $d[0, j]$ важи да су једнаки нули, тада горе описана рекурента веза важи и за елементе из прве врсте и прве колоне. Дакле, у овом случају базу можемо посматрати и преко овог дела матрице. Овим је имплементација доста поједностављена.

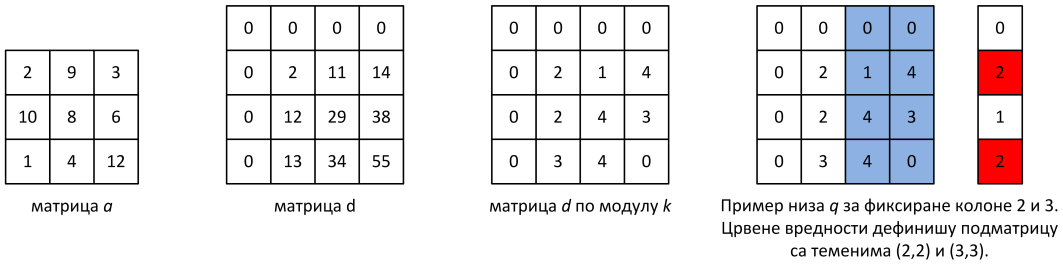
На описани начин добијамо алгоритам у сложености $O(n^4)$. Заиста, за сваку подматрицу суму рачунамо у константом времену, а њих има управо $O(n^4)$. Овде вршимо итерацију по свим могућим теменима (A_x, A_y) и (B_x, B_y), где је A горње лево а B доње десно теме подматрице (односно $A_x \leq B_x$ и $A_y \leq B_y$). Овај алгоритам доноси 30 бодова.

Оптимално решење добијамо уколико мало паметније рачунамо које матрице имају суму дељиву са k. Тачније, нама није битно које су то подматрице (а ми их управо и налазимо горе описаним алгоритмом) већ само колико их има. Наравно, уколико је потребно и излистати ове подматрице, већ описани алгоритам је оптималан (јер у "најгорем случају" сума сваке подматрица може бити дељива са k).

Фиксирајмо вредности A_y и B_y (у крајњем алгоритму ћемо итерирати по свим паровима). Да ли можемо "брзо" наћи број подматрица са овим условом чија је сума дељива са k? Другим речима, за колико парова A_x и B_x су суме подматрица дефинисаних овим теминима дељиве са k? Посматраћемо још један помоћни низа q. Вредности елемената низа q дефинишемо као (подсетимо се да смо вредности A_y и B_y фиксирали)

$$q[i] = \text{сума елемената подматрице са теменима } (1, A_y) \text{ и } (i, B_y)$$

Одавде добијамо да за пар A_x и A_y сума подматрице једнака $q[A_y] - q[A_x - 1]$ (овде узимамо да је $q[0] = 0$). Нама је потребан број парова за који је ова разлика дељива са k. Како је разлика два броја дељива са k ако и само ако оба броја дају исти остатак при дељењу са k, добијамо еквиваленту форму пребројавања парова: број парова елемената низа q који дају исти остатак при дељењу са k.



Слика 2. Анализа примера са папира.

Сада се већ лакше дише. Наиме, како је $k \leq 1.000.000$ можемо користити додатни низ num којим ћемо "бројати" елементе низа q са датом вредношћу. Дакле, $num[i]$ је једнак броју елемената низа q који имају остатак i при дељењу са k. На почетку иницијализујемо вредности низа num на нула, осим елемента $num[0]$ који постављамо на један. Овај елемент се односи на суму $q[0]$. Када обрађујемо елемент $q[i]$ гледамо претходне елементе који имају исти остатак. Ови елементи, са елементом i, управо граде наше парове. Дакле, резултат повећавамо за $num[q[i] \ mod \ k]$. Након овога и саму вредност $num[[i] \ mod \ k]$ повећавамо за један.

Алгоритам: Псеудо код оптималног решења проблема Шљивик

```
Input: матрица a димензије n × m и природни број k
Output: тражена број подматрица чија је сума дељива са k
постави матрицу d на нула матрицу;
for i ← 1 to n do
    for j ← 1 to n do
        d[i][j] = a[i][j] + d[i − 1][j] + d[i][j − 1] − d[i − 1][j − 1];
    end
end
toReturn = 0;
постави низ q на нула низ;
for x ← 1 to m do
    for y ← x to m do
        num[0] = 1;
        for i ← 1 to n do
            q[i] = (d[i][y] − d[i][x − 1]) mod k;
            toReturn = toReturn + num[currentSum];
            num[currentSum] = num[currentSum] + 1;
        end
        постави не нула елементе низа num на нуле;
    end
end
return toReturn
```

У овом алгоритму, након сваког фиксираног пара, потребно је низ num вратити на нула низ. Како је ограничење за дужину овог низа 1.000.000, тривијалан пролазак кроз низ и постављање сваког елемента на нулу није довољно брзо. Међутим како је број промењених

елемената највише $n \leq 400$, можемо једноставно поново проћи кроз посматране вредности из низа q и само њих поставити на нуле.

Овде треба бити пажљив са ограничењима. Како је нама увек потребно да знамо само остатак при дељењу са k, све посматране вредности (елементи низа q и матрице d) рачунамо по модулу k. Корисно је користити "вештачки" модуо, јер имамо само сабирања а не и множења. Под вештачким подразумевамо да користимо *while* петљу где посматрани број умањујемо (или увећавамо) за k све док не добијемо вредност из сегмента $[0, k-1]$. Напоменуто сабирање је битно јер ће у том случају *while* петља направити само пар итерација (ово не важи за елементе матрице a за које треба користити функцију *mod*).

Иницијализација матрице d има сложеност $O(n^2)$, јер сваки елемент иницијализујемо у константном времену. За сваки фиксирани пар y координата, којих има $O(n^2)$, рачунамо вредности низа q у линеарном времену. Рачунање број парова за x координату је такође линеаран, јер користимо низ num. Дакле, укупна сложеност овог алгоритма је $O(n^3)$.

РБ	Алгоритам n	Сложеност	Број бодова
01	Наивно решење	$O(n^6)$	10
02	Убрзано наивно решење матрицом сума d	$O(n^4)$	30
03	Бинарна претрага или брзо сортирање за низ q	$O(n^3 \log n)$	70
04	Оптимално решење	$O(n^3)$	100

Табела 1. Дистрибуција поена у односу на сложеност алгоритма.

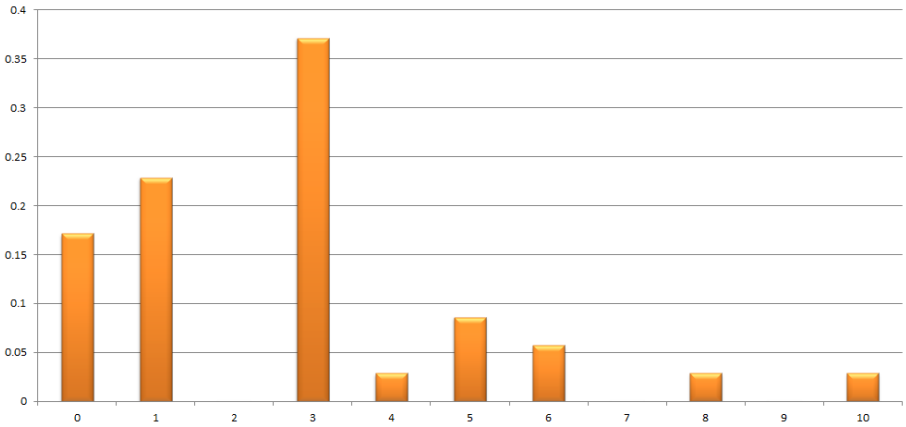
Напомена. Сличан проблем био је на Републичком такмичењу 2007. године - проблем Викендица (други проблем за Б категорију). Потребно ја наћи број подматрица чија сума припада датом сегменту $[A, B]$. Алгоритам фиксира горњи леви угао и испитује могуће подматрице (односно могућности за доњи десни угао). Слично проблему Шљивик, овде се крећемо по могућим ширинама (што је еквивалентно фиксирању две колоне). Треба приметити да, пошто баратамо са сумама, подматрице са ширином k добијамо преко подматрица ширина $k-1$ (јер је сума растућа функција). За рачунање сума користимо исти низ d. Сложеност и овог алгоритма је $O(n^3)$. Напоменимо да се проблем може урадити и преко бинарне претраге (уколико не користимо особину да је сума растућа).

Додатак А: Занимљивости из кодова такмичара

При тестирању решења такмичара, као и при при писању овог материјала, аутори су наишли на доста занимљивих ствари у кодовима. Овде су издвојене неке од њих. Наравно, суштина додатка је да се читалац насмеје и не треба га схватити као прозивање или исмевање (неке ствари су и за похвалу).

- **АПП приступ**: АПП приступ, који је заправо скраћеница од "ако прође - прође"[1], је заснован на следећем: уколико такмичар нема времена или не уме да реши неки од проблема, у излазну датотеку исписује неки резултат, у зависности од формата излаза, и нада се да ће неки од тест примера да прође. Ово је сасвим легални начин да се прикупе бодови, међутим тест примери се тако генеришу да би се ово избегло (примера ради уколико је резултат проблема бинарна вредност у улазу се заправо задаје више примера како би излаз био низ бинарних вредности).

  Како је било доста оваквих приступа, природно се намаће питање какву расподелу имају ове вредности. На слици испод је приказан резултат.



Слика 1. Расподела вредности при АПП приступу.

- **Грешка при АПП приступу:** Било је пар грешака (чак) и у имплементацији АПП приступа. Овде приказујем један тип грешке.

```
// Problem RECI
#include<stdio.h>

int main()
{
    FILE *out;
    out = fopen("lift.out","w");
    fprintf(out,"0");
    fclose(out);

    return 0;
}
```

- **Занимљиви коментари:** Издвајамо пар коментара на које смо наишли:

```
//system("PAUSE"); //          <---      TODO: ukloniti!!!

//moze i hash, ako ostane vremena

//#pragma omp parallel for
```

[1]Назив ове "технике" је оргиналан и први пут се појављује у овом документу.

```
 * Complexity: l - maximum needle length, L - maximum haystack length then complexity is O(mn(l + L))
 *
 * Algorithm:
 * Since the limits in this problem are very discrete, we need not do any heavy optimization here.
 * What we basically do is for each word in the list ("haystacks") we take each of the pre-input
 * words ("needles"), find all of its occurences in the haystack and cross them out. Then, we
 * iterate through all of the characters in the haystack and if all are crossed, we increment the
 * final result. Searching for occurences is accomplished using handy std::string::find, which uses
 * a worst case O(l + L) algorithm (it kepng a table of occurences for all characters in the needle
 * and can always  eliminate at maximum l characters from the search, yielding final complexity of O(l + L)).
```

- **Занимљива географија:** Како би показали да и комисија може да погреши, ево дела приче са *facebook* странице такмичарске комисије.

```
Takmicar: Zasto nisu postavljeni rezultati Severnobanatskog okruga (ucenika koji su izradili zadatke u Adi)?

Andreja Ilic:  hmm... to nije siglo koliko vidimo, ali u svakom slucaju ADA nije dozvoljena na nasim
takmicenjima: http://www.yuoi.nis.edu.rs/pravilnik.html#zadaci_programskookruzenje

Takmicar: To je optina u Srbiji! U blizini Kikinde, Sente, Madjarske granice... Tehnicka kola u Adi
vec 10 godina organizuje okruzno takmicenje iz programiranja, do sada nikad nije bilo problema.

Andreja Ilic: Aha mozda je doslo do zabune... Dakle, da li ovde pricao o Adi kao o programskom jeziku
ili o Adi kao o opstini :)
```