V НАЦИОНАЛЕН ЕСЕНЕН ТУРНИР ПО ИНФОРМАТИКА И ИНФОРМАЦИОННИ ТЕХНОЛОГИИ "ДЖОН АТАНАСОВ"

Секция "Информатика"
Условия и решения

ШУМЕН, 18-20 Ноември 2005

Организатори:

Министерство на
образованието и науката
Шуменски университет
“Епископ Константин Преславски”
Съюз на математиците в България
Школа А&Б – Шумен

Спонсори:

ICON - Шумен
“СтеЛи&Ко” – Шумен
“Еленко Компютърс” ЕООД Шумен
“Телепол” – Шумен
Книжарница “Словестност”
Община Шумен

Задача А1. Стари карти

Картите, които не са в електронен вид вече се смятат за стари. Те съдържат вярна и може би пълна информация, но за какво може да се използва карта, която дори няма търсачка? А без автоматично намиране на пътища, картата е напълно неизползваема за съвременните потребители. За нещастие, старите карти представят данните в много неудобен вид на огромна картина, която е пълна с излишна и неизползваема информация. Единствената съществена информация са градовете и пътищата между тях. Известно е, че между всеки два града съществува или директен път, или път, минаващ през няколко други града. Всеки път има фиксирана дължина, като между два града може да има най-много един пряк път. Ако такъв съществува в едната посока, то съществува и пряк път със същата дължина в другата посока.

Единствената надеждна информация, която може да се вземе в електронен вид от стара карта, е таблица с минималните разстояния между всеки два града. Градовете са безкрайно малки и когато преминавате през град, се счита, че изминавате 0 км.

Входните данни започват с ред, съдържащ числото N – броя на градовете ($3 \leq N \leq 500$). На следващите N реда има по N числа. j-тото число на i-тия ред е разстоянието от град i до град j и нека го означим с Dij ($1 \leq i, j \leq$ N). Знаем, че $0 < Dij = Dji \leq 1000000$. Също така $Dii = 0$.

Вашата програма **MAPS** трябва да отпечата описание на всички преки пътища, съществуващи на картата, ако е известно, че техният брой е минимален. Описанието на всеки път трябва да бъде отпечатано на отделен ред и трябва да съдържа три числа, разделени с интервал – двата града, свързани с пътя и разстоянието между тях. Тъй като старата карта е надежден източник на информация – задачата винаги ще има решение.

Примерен вход 1:
```
3
0 10 20
10 0 30
20 30 0
```

Примерен изход 1:
```
1 2 10
1 3 20
```

Примерен вход 2:
```
3
0 10 20
10 0 25
20 25 0
```

Примерен изход 2:
```
1 2 10
1 3 20
2 3 25
```

Решение:

Ако разгледаме градовете като върхове, а пътищата като ребра, можем да решим задачата в термините на крайни неориентирани графи.

В задачата разполагаме с матрица на разстоянията между всяка двойка върхове в граф G, който обаче е неизвестен. Нека означим тази матрица с $D(G)$. От нея елементарно може да се построи пълен граф P, чиято матрица $D(P) = D(G)$, като между всеки два върха поставим ребро с дължина равна на разстоянието между тях в матрицата. Първо забелязваме, че G е подграф на P – т.е. те имат едни и същи върхове и ако едно ребро е от G, тогава то е и от P. Това е така, защото за ребро от G, (a, b), свързващо върха a с върха b е изпълнено, че разстоянието между двата върха е точно равно на дължината на това ребро и следователно това е ребро и от P (ако допуснем, че реброто е по-късо, то разстоянието би било по-малко, а ако допуснем, че реброто е по-дълго, тогава то би било излишно и това противоречи с минималния брой ребра в G).

Нека разгледаме алгоритъм на Дейкстра в пълния граф P с начален връх i. На всяка стъпка от алгоритъма избираме оня връх j, за който разстоянието от i до j е най-малко измежду разстоянията от i до върховете, които не са избирани досега. Заедно с такъв връх j, ние избираме и ребро, което е последното в най-късия път от i до j. Нека на всяка стъпка се стремим това ребро да е минимално, т.е. ако има няколко пътя до j, да избираме този, за който последното ребро е минимално. Съвкупностите от тези ребра образуват покриващо дърво на графа P, което ще означим с $D(i)$.

Твърдим, че графа G е обединение на дърветата $D(i)$ за всеки връх i.

Доказателство: Тъй като върховете на дърветата и G съвпадат, разглеждаме само ребрата. Нека реброто $r(a, b)$ е от $D(i)$ за някое i и дължината му е len. Тогава разстоянието между a и b е равно на len (дължината е по-малка или равна на len, но не може да е по-малка, тъй като пътят до b би бил по-къс, без да се включва $r(a, b)$). Тогава върховете a и b в G са свързани или с $r(a, b)$, или с някакъв друг път със същата дължина. Но ако са свързани с път, то последното ребро в този път е по-късо от $r(a, b)$, което е противоречие с минималността на последното ребро за пътя от i до b.

Нека $r(a, b)$ е ребро от G, с дължина $lenr$. Нека разгледаме $D(a)$. Реброто $r(a, b)$ или е ребро от $D(a)$, или в $D(a)$ има друг път между a и b. Нека допуснем, че има друг път a, $x1$, $x2$, …, b с дължина len. От това, че $D(a)$ е построено с алгоритъм за минимална дължина от a до всеки връх следва, че $len \leq lenr$. Но от първата част на доказателството всяко от ребрата $r(a, x1)$, $r(x1, x2)$, ... $r(xn, b)$ е от G. Тогава в G има път с дължина len ($len \leq lenr$), свързващ a и b. Тогава реброто $r(a, b)$ е излишно, защото не променя разстоянието между a и b в G, както и кое да е друго разстояние в G. Това е в противоречие с минималния брой ребра в G. Оттук следва, че $r(a, b)$ участва в $D(a)$.

С това доказахме, че че графът G е обединение на дърветата $D(i)$ за всеки връх i. В доказателството съществено използвахме и факта, че дължините на ребрата са положителни.

Задача А2. Цифри

Дадено е число N, записано в K-ична бройна система, $2 \leq K \leq 36$. Главните латински букви A, B, C, D, E, F,… означават цифри, съответно със стойности 10, 11, 12, 13, 14, 15, ... Дадена е и една цифра M (число между 0 и 9, или главна латинска буква, такава че цифрата която тя означава е по-малка от K). Да се напише програма **DIGITS**, която намира колко пъти цифрата M се използва за записването на всички числа от 1 до N в K-ична бройна система.

На първия ред от входния файл са записани числото R ($1 \leq R \leq 1000000$) (което задава дължината на числото N), цифрата M (записана с десетично число от 0 до $K - 1$) и числото K, всичките разделени с интервал. На следващия ред се намира числото N, записано в K-ична бройна система. То се състои от последователни цифри, без интервал между тях.

На единствения ред в изходния файл програмата трябва да изведе търсения брой, записан в K-ична бройна система.

В около 40% от тестовете M ще е равно на 0.

В около 20% от тестовете $R \leq 6$.

В около 50% от тестовете $R \leq 5000$.

Примерен вход:
```
6 11 36
GWFERZ
```
Съответният изход:
```
3CMATS
```

Примерен вход:
```
6 0 10
107351
```
Съответният изход:
```
49517
```

Задача А3. Американска рулетка

Студентите Тошо и Гошо дълго се чудили как да се присмеят на приятеля си Пешо хакера, който не знаел че 36, а не 49, е най-голямото число, на което може да спре топчето в Американската Рулетка. Накрая им дошла гениалната идея да играят следната игра. За целта нашите герои имат нужда от тесте от карти, в което да има по 4 карти с номера 1, 2, 3, 4, 5 и 6, т. е. тесте с 24 карти. Играта протича по следния начин. Разпръскват картите на една маса, така че да могат да виждат стойностите им и, редувайки се, избират по една карта от масата и я хвърлят на земята. Ако даден играч не може да избере карта, така че като я хвърли на земята, сумата от стойностите на всички карти на пода да е не повече от 49, то той губи.

Главната цел на Тошо и Гошо била те да играят играта докато Пешо влиза в стаята, той да ги попита за правилата и те да се пошегуват с него. Така и станало, но Пешо, понеже е хакер, има един скрит коз. Той би могъл с ваша помощ да покаже на Тошо и Гошо, че всъщност е много умен, като им каже кой може да спечели вече започната игра и кой е печелившият ход. Ако помогнете на Пешо да отговори бързо на горните въпроси, той ще възвърне уважението на своите приятели. За целта напишете програма **ROULETTE**, която да има следния вход и изход:

Вход. Влизайки в стаята, Пешо заварва една вече започната игра. Той ви подава на първия и единствен ред на стандартния вход картите на земята в реда, в който са били хвърлени. Например 335 значи, че първо Гошо е хвърлил 3, после Тошо е хвърлил 3 и накрая пак Гошо е хвърлил 5. Сега на ред е Тошо. Винаги първи играе Гошо.

Изход. На първия ред на стандартния изход се очаква вие да кажете кой ще победи, като изведете една от латинските букви ‘T’ или ‘G’, съответно за Тошо и Гошо. Също така трябва да изведете и каква карта трябва да изиграе този, който е наред, за да спечели, или 0, ако той няма никакъв шанс да спечели при оптимална игра на противника. Ако съществуват няколко печеливши хода, изведете този, при който се играе най-голяма карта.

Примери

Вход
```
355
```

Изход
```
T 6
```

Вход
```
44223553556
```

Изход
```
G 0
```

Вход
```
1221
```

Изход
```
G 6
```

Решение:

В задачата се иска да се реши една проста комбинаторна игра. Позиция в играта ще наричаме броя от останалите карти от всеки вид и това кой е наред. Тъй като за всеки вид карти може да имаме останали на масата карти цяло число между 0 и 4 в ключително, това прави 5 ситуации за всеки вид карти. Понеже имаме 6 вида карти, то значи можем да съпоставим на всяка ситуация от останали карти на масата число между 0 и $5^6-1 = 15624$, вкючително. Трябва да имаме и в предвид кой е на ход в момента и, след като имаме двама играчи, то всяка Позиция може да се кодира с число между 0 и 31249. Когато позициите на една игра се кодират с числа в достатъчно малкък интервал и можем да заделим масив с такава дължина, казваме, че играта е обхватна.

Играта, която играем, също така е ациклична. Тоест, от дадена Позиция след извършването на x ($x > 0$) хода, не можем да се върнем пак в същата Позиция, тъй като всеки ход намалява сумата от стойностите на всички карти на масата.

Печеливша Позиция в играта е такава Позиция, от която съществува ход, с който отиваме в губеща Позиция (тоест пращаме противника в тази губеща Позиция).

Губеща Позиция е такава от която или всички ходове ни водят до печеливша Позиция, или нямаме възможни ходове.

От направените дотук расъждения забелязваме, че можем да напишем рекурсивна процедура, която за дадена Позиция да сметне дали тя е печеливша или губеща. За целта трябва да знаем за всички позиции, до които се стига с едни ход, дали са губещи или печеливши, което можем да направим, като извикаме рекурсивната процедура за тях.

За да изведем кой е печелившият ход, трябва просто в масив да помним за всяка печеливша Позиция кой ход я превръща в губеща.

Задача B1. Две и три

Разполагаме с неограничен брой десетични цифри 2 и 3. С част от тях искаме да напишем десетично число, което да се дели на 2^n, където n е естествено число. При това търсим най-малкото такова число.

Ако например $n = 4$, една цифра не стига да напишем такова число: 2 е четно, но не се дели на $2^4 = 16$, 3 пък направо е нечетно. С две цифри от наличните могат да се напишат (по големина) 22, 23, 32 и 33. Числото 32 вече има нужното ни свойство (впрочем, то се дели даже и на $2^5 = 32$). Факт е, че например 33232 също изпълнява условието, но то е много по-голямо. Няма изискване и двете цифри да се срещат задължително в записа, нито пък да са с равен брой срещания.

Напишете програма **N23**, която намира желаното число или установява, че такова не съществува.

От стандартния вход се въвежда един ред, съдържащ естественото число n, $1 \le n \le 10000$.

На стандартния изход програмата трябва да изведе един ред с намереното най-малко естествено число, в десетичния запис на което има само цифрите 2 и 3 (по колкото е необходимо от всяка от тях) и което се дели на 2^n; или един ред със съобщението NO, ако такова число не съществува.

Пример. Вход:
```
11
```

Изход:
```
223232
```

Решение

Още от основния училищен курс за десетично записаното естествено число m е известно, че:

- дели се на 2, ако е четно (т.е. – последната му (*една*) цифра се дели на 2);
- дели се на 4, ако числото, образувано от последните му *две* цифри се дели на 4 (ако е едноцифрено, можем да считаме, че има водеща нула);
- дели се на 8, ако числото, образувано от последните му *три* цифри се дели на 8 (отново можем да дописваме водещи нули, ако няма достатъчно цифри).

Обобщението на тези правила се очаква и лесно се доказва: $\boldsymbol{m}$ се дели на 2^n тогава и само тогава, когато числото, образувано от последните му $\boldsymbol{n}$ цифри в същия ред се дели на 2^n (ако цифрите на $\boldsymbol{m}$ не стигат, можем да си мислим, че има водещи нули).

Като вземем предвид и елементарното съображение, че щом $\boldsymbol{m}$ се дели на 2^n, то се дели и на всички по-малки степени на двойката, заключаваме, че с ограничения ресурс от десетични цифри трябва да изграждаме „опашки" от $\boldsymbol{n}$ цифри, гарантиращи делимостта на 2^n. В нашия случай се оказва, впрочем, че такава „опашка" е еднозначно определена. Наистина – последната цифра задължително е 2, иначе пропада делимостта на 2; предпоследната – задължително 3, което осигурява делимост на 4; следващата по старшинство – задължително 2, иначе няма делимост на 8 и т. н.

Алгоритъмът за изграждане на тази редица се оказва елементарен, което еднозначно ни осигурява число, записано с общо n на брой цифри (двойки и тройки), което се дели на 2^n, т.е. задачата винаги има решение с не повече от n цифри. По-интересно е изискването да се намери най-малкото решение. Наистина, 23232 (което е опашката от пет цифри) се дели на 2^5=32, но още 32=00032 има това свойство (и се записва само с две цифри).

Алгоритъм за получаване на опашката. Да разгледаме сега по-подробно аритметиката на получаване на редицата $\{\boldsymbol{c_i}\}$ от цифри в „опашката", започвайки от най-младшата – $\boldsymbol{c_1}$. Единствената възможност, както споменахме, е $\boldsymbol{c_1}$=2. Всяка от следващите (по старшинство) цифри от „опашката" m трябва да осигурява делимост на още една степен на двойката. На i-тата стъпка ($i > 1$) ще имаме $\boldsymbol{c_i}$=2 или $\boldsymbol{c_i}$=3, т. е., ако означим с

$$m_i = \overline{c_i \underbrace{c_{i-1} c_{i-2} \dots c_1}_{m_{i-1}}}$$

естественото число, записано с тези цифри в този ред, имаме:

$$m_i = \begin{cases} 2.10^{i-1} + m_{i-1} \\ \textit{или} \\ 3.10^{i-1} + m_{i-1} \end{cases}$$

като изборът е еднозначно определен от това, дали „досегашната опашка" m_{i-1} вече се дели и на 2^i (на 2^{i-1} тя задължително се дели) или не. Наистина, ако $m_{i-1} = 2^i.k$, където $\boldsymbol{k}$ е естествено число, единствената възможност да получим още един делител на 2 е да вземем първата алтернатива:

$$m_i = 3.10^{i-1} + m_{i-1} = 3.2^{i-1}.5^{i-1} + 2^{i-1}.t = 2^{i-1}(3.5^{i-1} + t)$$

(не втората, защото тогава първото събираемо не се дели на 2^i, второто се дели → сумата не се дели). И точно напротив – при $m_{i-1} \neq 2^i.k$ ще имаме $m_{i-1} = 2^{i-1}.t$, където $\boldsymbol{t}$ е нечетно. Единствена възможност сега остава

$$m_i = 2.10^{i-1} + m_{i-1} = 2.2^{i-1}.5^{i-1} + 2^i.k = 2^i(5^{i-1} + k)$$

като числото в скобите е четно (сума от две нечетни) и, следователно, произведението се дели на 2^i. По аналогични на предишния случай причини другата алтернатива не може да бъде избрана. Това доказва еднозначността и показва алгоритъма за получаване на редицата от цифри $\{\boldsymbol{c_i}\}$.

Алгоритъм за решаване на задачата. Въз основа на горните разглеждания, естественият подход към атакуване на проблема е да изграждаме опашката по указания алгоритъм и да спрем в първия момент, когато получим делимост на 2^n. Малко запомняне на предишни резултати ще ускори изчислителния процес.

1. Въвежда се естественото число n.
2. Инициализираме променливи: резултат: res = 2; степен на петицата: deg5 = 1; изчислена база за пресмятане на новата стойност k = 1; осигурена степен на двойката, на която res се дели: deg = 1.
3. Проверяваме дали deg е по-малко от n. Ако да – към т. 4 иначе към т. 11.
4. Ако k е четно, долепваме към res водеща цифра 2, иначе – водеща цифра 3.
5. Минимално осигурената степен на делимост сега е равна на броя на цифрите на res, затова deg приема тази стойност.
6. Проверяваме дали deg все още е по-малко от n. Ако да – към т. 7 иначе към т. 11
7. Намираме следващата степен на петицата: deg5 := 5*deg5
8. k :=((новодобавената цифра)*deg5+k) / 2
9. Проверяваме каква степен на двойката дели k, като увеличаваме deg с толкова. Ако достигнем (или надхвърлим) **n**, към т. 11.
10. Към т. 4.
11. Извеждаме резултата res.

Примерна реализация

```
#include <stdio.h>
int n;
typedef struct {int count;
                char dig[10001];
               } Long;

Long res={1,{2}},deg5={1,{1}},k={1,{1}};

void Long_Add(Long *a, Long *b, Long *r)
{char carry=0;
 int i;
 r->count=(a->count>b->count)?a->count:b->count;
 for (i=0;i<r->count;i++)
 {if (i<a->count&&i<b->count) r->dig[i]=a->dig[i]+b->dig[i];
  else r->dig[i]=(i<a->count)?a->dig[i]:b->dig[i];
  r->dig[i]+=carry;
  if (r->dig[i]>9) {r->dig[i]-=10;carry=1;}
  else carry=0;
 }
 if (carry) r->dig[r->count++]=1;
}
```

```
void Long_MulDig(Long *a,char d,Long *r)
{char carry=0;
 int i,t;
 r->count=a->count;
 for(i=0;i<a->count;i++)
 {t=d*a->dig[i]+carry;
  r->dig[i]=t%10;
  carry=t/10;
 }
 if (carry) r->dig[r->count++]=carry;
}

int Long_Div2(Long *a,Long *r)
{int i,d=0;
 r->count=a->count;
 for (i=a->count-1;i>=0;i--)
 {d=10*d+a->dig[i];
  r->dig[i]=d>>1;
  d&=1;
 }
 if (!r->dig[r->count-1]) r->count--;
 return d;
}

void Long_Show(Long *a)
{int i;
 for (i=a->count-1;i>=0;i--) printf("%d",a->dig[i]);
 printf("\n");
}

int main(void)
{int deg=1;
 Long t;
 scanf("%d",&n);
 while (deg<n)
 {res.dig[res.count++]=2+(k.dig[0]&1);
  deg=res.count;
  if (deg==n) break;
  Long_MulDig(&deg5,5,&deg5);
  Long_MulDig(&deg5,res.dig[res.count-1],&t);
  Long_Add(&t,&k,&t);
  Long_Div2(&t,&k);
  t=k;
  while (deg<n&&!Long_Div2(&t,&t)) deg++;
 }
 Long_Show(&res);
 return 0;
}
```

Задача B2. Папки

Хакер Ш. имал два компютъра – A и B. На твърдия диск на A той работел в папка с дърво от подпапки в нея, където били записани файловете му. От време на време Ш. преписвал на твърдия диск на B цялата своя папка от A и така правел back-up. На компютъра B нищо друго не се правело. Но с течение на времето файловете му ставали все повече и той решил да копира само тези от тях, които са по-нови в сравнение с вече копираните от предишния път. Разбира се, ако е направил нова подпапка, копирал я цялата, заедно с всичките нейни подпапки и файлове.

Напишете програма **FOLDERS**, която при зададено описание на папките в A и B, пресмята броя на файловете, които трябва да се копират.

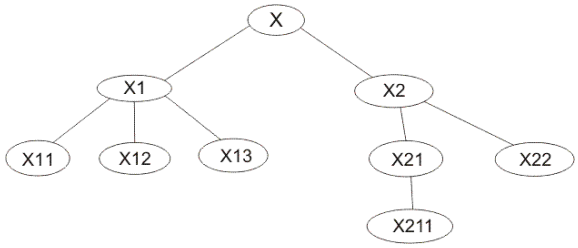
Входните данни се четат от стандартния вход. Дървовидната структура на папките е зададена, започвайки с главната папка, чието име е написано в първия ред на входния файл (низ от малки латински букви и цифри), слевано от броя на намиращите се в нея подпапки и/или файлове и времето на създаването й (в секунди, от някакъв начален нулев момент). На следващите редове са нейните подпапки и файлове, като в случая, когато е даден файл, броят на съдържащите се в него подпапки и файлове е отбелязан с 0, а даденото време се отнася за момента на последната промяна във файла. Изобщо казано, входният файл е образуван, като първо е записан коренът на дървото, след него – неговите наследници. След това, след всеки наследник се вмъкват редове за неговите наследници (ако има такива) и така нататък, докато се изчерпят всички наследници. След като се запише файловата структура на A, във входния файл се записва по аналогичен начин файловата структура на B.

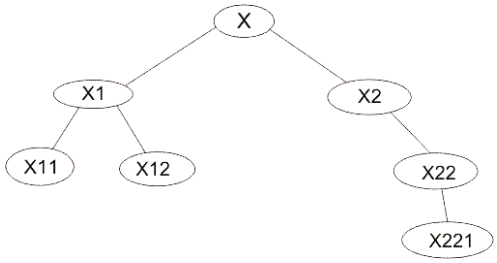
Програмата трябва да изведе броя на файловете, които трябва да бъдат копирани от A в B.

Пояснения и ограничения: Няма еднакви имена в една папка. Няма празни папки. Ако две папки (или файлове), едната намираща се в A, а другата – в B, са с еднакви имена и са на едно и също съответно място в йерархията, но папката (файлът) в A е създадена по-късно от тази в B, копира се цялата й система от подпапки и файлове (ако е файл – копира се само файлът, като той заменя съответната папка в B, която се изтрива). Максимален брой редове във входния файл: 200. Максимален брой подпапки и/или файлове в една папка: 9. Максимална дължина на името на файл или папка: 9 Максимален момент от времето 9999 сек.

Пример. Вход:

```
x 2 0
x1 3 1
x11 0 14
x12 0 11
x13 0 13
x2 2 5
x21 1 20
x211 0 21
x22 0 16
x 2 0
x1 2 1
x11 0 9
x12 0 11
x2 1 5
x22 1 6
x221 0 7
```





Изход:

```
4
```

Решение:

Структурата данни, в която се пази дървото на папките от А използва масивите `int a[N][M], at[N]; char as[N][10];` В `at[j]` е записан съответният момент време за `j`-тата папка (или файл), в `as[j]` – името й. Броят на наследниците й се съхранява в елемента `a[j][0]`, а номерата на наследниците – в `a[j][1]`, `a[j][2]`, ..., `a[j][a[j][0]]`. В `na` е общият брой на всички папки и файлове от А. По аналогичен начин се изгражда структурата от данни за В, като се ползват съответно `b[N][M], bt[N], bs[N][10]` и `nb`. Запълването на тези данни се извърша от фукцията create(), извиквана два пъти, за да обработи входния файл за А и за В.

Функцията `compare(int n)` рекурсивно и синхронно обхожда двете дървовоидни структури, спускайки се в дълбочина. Всеки път, когато се достигне връх на дървото А, при което се открива, че съответния връх на В няма необходимите свойства за "еднаквост" (например върхът в А е папка, а този в В – е файл, или двата върха са файлове, но този в А е с по-ново време), навлизането в дълбочина се прекратява и евентуално се извиква рекурсивната функция `count()`, която преброява файловете от поддървото на А, което започва от въпросния връх. Променливата `c` служи за глобален брояч, чиято стойност се отпечатва накрая на програмата. Във функцията `main()` се обработва коренът на дървото, който има номер 1.

```
//ANSI C++
#include<iostream>
#include<cstring>
using namespace std;

const int N=999;
const int M=19;
int a[N][M], at[N]; char as[N][10];
int b[N][M], bt[N]; char bs[N][10];
int n,na,nb;
int c=0;

void create(int a[N][M], int at[N], char as[N][10])
{
 cin >> as[n] >> a[n][0] >> at[n];
 int n0=n;
 for(int i=1; i<=a[n0][0]; i++)
  {
    n++;a[n0][i]=n;
    create(a, at, as);
  }
}

void count(int n)
{
 if(a[n][0]==0) c++;
 else
 for(int i=1;i<=a[n][0];i++) count(a[n][i]);
}

void compare(int n)
{
 int i,j,j0;
 for(i=1; i<=a[n][0]; i++)
  {
    int f=0;
    for(j=1; j<=b[n][0]; j++)
```

```
     if(strcmp(as[a[n][i]],bs[b[n][j]])==0) {f=1; j0=j;}
    if(f==1)
    {
     if(at[a[n][i]]==bt[b[n][j0]])
      {
        if((a[a[n][i]][0]!=0)&&(b[b[n][j0]][0]!=0)) compare(a[n][i]);
        else if((a[a[n][i]][0]==0)&&(b[b[n][j0]][0]!=0)) c++;
        else
         if((a[a[n][i]][0]!=0)&&(b[b[n][j0]][0]==0)) count(a[n][i]);
      }
     if(at[a[n][i]]>bt[b[n][j0]]) count(a[n][i]);
    }
    if(f==0) count(a[n][i]);
  }
}

int main()
{
 n=1;create(a, at, as);na=n;
 n=1;create(b, bt, bs);nb=n;
 if(strcmp(as[1],bs[1]) != 0) count(1);
 else if((a[1][0]==0)&&(b[1][0]!=0)) count(1);
 else if((a[1][0]!=0)&&(b[1][0]==0)) count(1);
 else if(at[1] > bt[1]) count(1);
 else compare(1);
 cout << c << "\n";
 return 0;
}
```

Задача B3. Зелени площи

В град Ш. има много зелени площи. Всяка от зелените площи е с формата на изпъкнал многоъгълник. За да се поддържат зелените площи в изряден вид, кметът на града решил да помоли Директора на Математическата гимназия в Ш. да възложи на всеки ученик поддържането на поне една от тях. Директорът пък решил да възложи на ученика от 9 клас с профил Информатика Шибил да направи програма, която по случаен начин да разпредели площите между учениците. Шибил, обаче, решил да се възползва от възможността и да остави за себе си най-малката площ. Понеже току що е започнал да учи програмиране, задачата му се сторила много трудна. Помогнете му, като напишете програма **GREEN**, която да намира зелената площ с минимално лице.

Входните данни ще бъдат зададени на стандартния вход. На всеки ред ще бъде зададено описанието на един от многоъгълниците. То започва с броя M на върховете на многоъгълника ($3 \le M \le 100$), последван от M -те двойки целочислени координати на върховете (в правоъгълна координатна система), зададени в посока на часовниковата стрелка. Всички числа са разделени с по един интервал. След описанието на последния многоъгълник следва ред, съдържащ само числото 0.

На стандартния изход програмата трябва да изведе номера на многоъгълника с най-малко лице, като номерацията на многоълниците започва от 1 и следва реда, по който те са зададени на входа. Ако два или повече многоъгълника са с едно и също минимално лице, да се изведе номера на този от тях, който има по-малко върхове. Ако два или повече многоъгълника са с едно и също минимално лице и един и същ брой върхове, да се изведе този от тях, който има по-малък номер.

Пример. Вход:

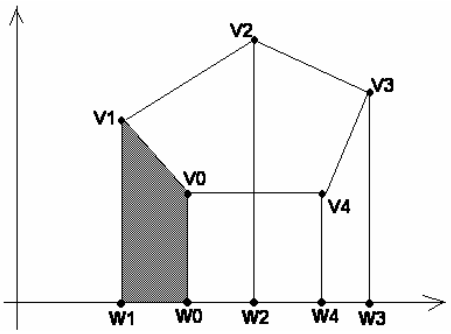
```
4 13 11 13 21 23 21 23 11
3 -15 -5 0 5 15 -5
4 -5 -5 -5 5 5 5 5 -5
0
```

Изход:

```
1
```

Решение:

Задачата за намиране на лице на многоъгълник е основна за изчислителната (комбинаторната) геометрия. Освен пряко по предназначение, намирането на лицето на многоъгълник може да се окаже важна стъпка при решаване на други интересни задачи. Преди да се спрем на авторското решение, нека да отбележим, че за намирането на лице на изпъкнал многоъгълник могат да се предложат различни алгоритми. Една възможност е да разбием многоъгълника с върхове V_0, V_1,…, V_{N-1} на триъгълници $V_0V_1V_2$, $V_0V_2V_3$,…, $V_0V_{N-2}V_{N-1}$, да намерим лицето на всеки триъгълник, например с използване на формулата на Херон и да пресметнем лицето на изпъкналия многоъгълник като сума от лицата на триъгълниците. Подобен подход има сериозен недостатък – изчисленията при такъв алгоритъм са свързани със загуба на точност. При формулата на Херон ще са ни необходими дължините на страните на триъгълника, при намирането на които ще се наложи използването на коренуване, а и при пресмятането на окончателния резултат ще се наложи да се коренува още веднъж. В изчислителната геометрия коренуването, както делението и пресмятането на функции като sin, cos, tg, arctg и т.н., често води до загуба на точност. Затова използването на такива пресмятания трябва да се избягва или да се сведе до минимум.



Фиг. 1

За решаване на задачата се предлага алгоритъм, който не предизвиква подобни изчислителни проблеми – т.н. *ориентирани лица*. Същността на алгоритъма е в следната формула:

$$S=|(x_1-x_0)*(y_1+y_0)/2+(x_2-x_1)*(y_2+y_1)/2+\ldots+(x_{N-1}-x_{N-2})*(y_{N-1}+y_{N-2})/2+(x_0-x_{N-1})*(y_0+y_{N-1})/2|$$

където (x_I,y_I), $I=0,1,\ldots,N-1$, са координатите на върховете на изпъкналия многоъгълник, в реда, в който се срещат при обхождането му (по посока на часовниковата стрелка или в обратна посока). На верността на тази формула няма да се спираме тук, а само ще отбележим, че стойността на израза $(x_I-x_{I-1})*(y_I+y_{I-1})/2$ е ориентираното лице на трапеца $V_{I-1}V_IW_IW_{I-1}$, където W_I и W_{I-1} са ортогоналните проекции на V_I и V_{I-1} върху абсцисната ос (вж. Фиг. 1 където е защриховано лицето на трапеца $V_0V_1W_1W_0$). Ориентирано лице е лицето на трапеца със знак плюс или минус в зависимост от това дали W_I е по-надясно от W_{I-1} върху абсцисната ос. Така ориентираното лице на защрихования на Фиг.1 трапец е отрицателно и при пресмятането ще бъде извадено от положителното ориентирано лице на трапеца $V_1V_2W_2W_1$.

Забележете още, че въпросната формула е валидна не само за изпъкнали многоъгълници, но и за неизпъкнали несамопресичащи се.

И така, за пресмятане на лицето на един многоъгълник не е необходимо да запомняме координатите на върховете му в паметта. Достатъчно е да помним само координатите на началната точка, за да пресметнем последното събираемо на израза. За намирането на минимума също не е необходимо да запомняме намерените лица.

На Фиг. 2 е показан програмен фрагмент с основните стъпки на алгоритъма.

```
minface=(double)MAX; minv=MAX; minn=MAX;
i=1;
while(1)
{
   cin>>M;
   if(M==0) break;
   face=0.;
   cin>>x1>>y1; x0=x1;y0=y1;
   for(j=2;j<=M;j++)
   {
     cin>>x2>>y2;
   face+=(x2-x1)*(y2+y1)/2;
   x1=x2;y1=y2;
   }
   face+=(x0-x1)*(y0+y1)/2;
   if(face<0) face= -face;
   if((face<minface)||(face==minface&&M<minv))
   {minface=face;minv=M;minn=i++;}
}
cout<<minn<<endl;
```

Фиг. 2

Задача С1. Хипервръзка

Чухте ли за новия browser? Pencho Browser е най-новото творение на Пенчо, с което той много се гордее. Той притежава всички качества, за да се превърне в абсолютен хит и, освен това, вече е почти време да се издаде първата му официална стабилна версия. Ръководството на проекта е дало срок до следващата седмица да има готова работеща версия, но за съжаление има още много бъгове за поправяне!

Вече почти няма време, а намирането на бъга, който изяжда хипервръзките в html документите, все още не е локализирано (това е едно от най-лошите наследства на бившия разработчик на проекта Унуфри). Тази част от кода е доста объркана и на всичкото отгоре е написана на асемблер и намирането на причината за това бъгче доста се усложнява. Поради тази причина, екипът на проекта реши да поправи сбърканите документи, като сложи отново хипервръзките в тях. Тъй като в момента всички от екипа са заети с това да усъвършенстват уменията си по Quake, вие сте тези, който трябва да се заемете с тази така важна задача.

Трябва да напишете програма **HLINKS,** която чете текст от стандартния вход (текстът може да е на няколко реда) и отпечатва този текст на стандартния изход**,** като търси всички хипервръзки в текста и за всяка намерена връзка LINK, замества LINK с (всичко се изписва с малки букви):

```
<a href="LINK">LINK</a>
```

Дефиницията на хипервръзка в тази първа версия на browser-а е доста проста. Тя отговаря на следните условия:

1. Съдържа само латински букви, цифри, точки (.) и наклонени черти (/);
2. Ако пред връзката има залепен (без никакви разделящи интервали или знаци) `http://`, този низ трябва да се прибави към тази връзка;
3. Във връзката трябва да има поне една точка;
4. Ако след най-дясната точка няма наклонени черти, буквите след тази точка трябва да са повече от 1 и по-малко от 5;
5. Преди и след всяка връзка, или няма нищо, или има знак различен от буква, цифра, точка и наклонена черта;
6. Никога връзка не може да започва с наклонена черта или точка;
7. Връзката не може да има 2 или повече последователни точки.

Знае се, че текстът не е по-дълъг 10000 знака и тези знаци са взети измежду знаците на ASCII таблицата. Текстът във входа завършва с края на файла.

Примерен вход:

```
Tova e link kym nai-noviq Pencho Browser:
http://www.pencho.browser.com, a
tova ne e: http://document.com/pencho.comma
I malko reklama - infoman.musala.com
```

Примерен изход:

```
Tova e link kym nai-noviq Pencho Browser:
<a href="http://www.pencho.browser.com">http://www.pencho.browser.com</a>, a
tova ne e: http://document.com/pencho.comma
I malko reklama - <a href="infoman.musala.com">infoman.musala.com</a>
```

Задача С2. Школа

Започна новата учебна година и в Школата по информатика в град Ш. отново се събраха много нови състезатели. За да подсигури редовното уведомяване на учениците за сбирките на групата, ръководителката на Школата направила малко проучване и установила, че новите ученици са N ($5 \le N \le 500$), номерирани с числата от 1 до N, по реда, в който са записани в дневника на Школата. M двойки от тях се познават и когато единият от двойката научи за поредната сбирка, може да уведоми другия за сбирката. За да си улесни работата, ръководителката би искала да избере колкото може по-малко ученици, които да уведоми лично, а те от своя страна да разпространят информацията до всички, като използват познанствата помежду си. Разбира се, че всеки уведомен за сбирката може да се свърже с всички свои познати и да ги уведоми, ако някой друг не го е направил до момента.

Напишете програма **SCHOOL**, която по зададени познанствата на учениците намира и извежда на стандартния изход минималния брой ученици, които трябва да бъдат уведомени от ръководителката така, че информацията да може да стигне до всички останали.

На първия ред на стандартния вход ще бъдат зададени числата N и M**,** разделени с един интервал. На всеки един от следващите M реда, разделени с интервал, са зададени два номера на ученици, които се познават.

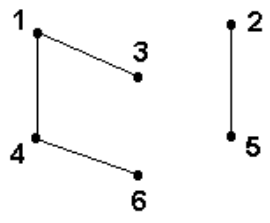
Пример.
Вход:
```
6 4
1 3
1 4
2 5
4 6
```

Изход:
```
2
```

Решение:

За решението на тази задача ще използваме техника от теорията на графите. Неориентираният ***граф*** $G(V,E)$ е съставен от множество от ***върхове*** $V=\{1,2,\ldots,N\}$ и множество от ***ребра*** E такова, че всяко ребро от E свързва два върха u и w от E – означаваме го с (u,w). Редицата от върхове $u_1,u_2,\ldots,u_k$ наричеме ***път*** от върха u_1 до върха u_k, ако всеки два съседни в редицата върха u_i и u_{i+1} са свързани с ребро. Когато $u_1= u_k$ тогава пътя наричаме ***цикъл***. Графът е ***свързан***, ако в него има път от всеки връх до всички останали. Графът е ***дърво***, ако е свързан и няма цикли. Дървото $D(V,E')$ такова, че $E' \subseteq E$ наричаме ***покриващо дърво*** на G. В сила е следното важно свойство: графът G е свързан тогава и само тогава, когато има покриващо дърво.

Ако графът не е свързан, тогава той се разпада на отделни свързани ***подграфи***, всеки от които наричаме ***свързана компонента***. На Фиг. 1 е показан граф с 6 върха и 4 ребра, който не е свързан и има 2 свързани компоненти (върховете 1, 3, 4 е 6 са в едната, а върховете 2 и 5 в другата).



Фиг 1.

	0	1	2	3	…	
1	2	3	4			
2	1	5				
3	1	1				
4	2	1	6			
5	1	2				
6	1	4				

Фиг. 2.

Да представим даденото в задачата като неориентиран граф. На всеки от учениците да съпоставим връх на графа и да свържем с ребро два върха, ако съответните им ученици се познават и, ако единият от тях бъде уведомен за сбирка на Школата, може да уведоми другия. Нека ученикът, съответен на върха с номер 1 е уведомен за сбирката, той може да уведоми всички ученици с които се познава. Всеки от тях може да уведоми всеки от познатите си, които още не са уведомени. В резултат ще бъдат уведомени всички ученици, за върховете на които в графа има път до (от) върха с номер 1, т.е. тези които са в една и съща свързана компонента на графа. Очевидно, за всяка свързана компонента ще е необходимо и достатъчно да бъде уведомен точно един от учениците, т.е търсеният в задача минимален брой ученици, които трябва да бъдат уведомени е равен на броя на свързаните компоненти на графа.

За намиране на броя на свързаните компоненти ще приложим сравнително универсалната техника "обхождане в ширина". С малка модификация, обаче, алгоритъмът може да се приложи и за решаване на задачата за определяне на самите компоненти (опитайте се да го направите сами).

Много важно за успешната реализация е начинът по който ще представим графа. Ще използваме представяне, което наричаме "списъци на съседите". Нека `g` е двумерен масив с толкова реда, колкото са върховете на графа и стълбове с 1 повече от върховете на графа (в реализацията използваме редовете с номера 1,2,..., *N* и стълбовете с номера 0,1,2,..., *N*). Списъкът на съседите на върха *u* съхраняваме в реда с номер *u*. Първият, вторият и т.н до *k*-тия съсед на *u* записваме в първия, втория и т.н. до *k*-тия елемент на реда, а в нулевия елемент поставяме броя *k* на съседите на *u*. На Фиг.2 е показано представянето на графа от Фиг.1 със списъци на съседите.

При "обхождане в ширина" използваме опашка `q`, като в променливите `qs` и `qe` се намират началото и краят на опашката. Започваме с произволен начален връх (например 1). Поставяме го в опашката, обявяваме го за обходен (като поставим 1 в съответния елемент на масива `used`), преброявяме първата свързана компонента (променливата `cnt`) и обхождаме тази компонента в ширина. За целта, докато в опашката има елементи, изваждаме елемент от началото на опашката и добавяме в края на опашката всички негови необходени съседи, обявявайки всеки един такъв съсед за обходен. Ако в края на тази стъпка няма необходени върхове – алгоритъмът завършва. Ако имаме необходен връх – избираме този връх за начален, преброяваме още една компонента и повтаряме описаните вече стъпки. На Фиг. 3 е показан програмен фрагмент с основните стъпки на алгоритъма.

```
cnt=0;
for(i=1;i<=N;i++)  {used[i]=0;g[i][0]=0;}
for(k=1;k<=N;k++)
{
   if(used[k]==1) continue;
   qs=qe=0;q[qs]=k;used[k]=1;cnt++;
   while(qs<=qe)
   {
      x=q[qs++];
      for(i=1;i<=g[x][0];i++)
      {
         y=g[x][i];
       if(used[y]==0)
       {used[y]=1;q[++qe]=y;}
    }
   }
 }
cout<<cnt<<endl;
```

Фиг. 3

Задача С3. Числа

Дадени са 3 двуцифрени числа d_1, d_2, d_3.

Напишете програма **NUMBER**, която намира броя M на n-цифрените числа $a_1\ a_2 \ldots a_n$, за които всеки две съседни цифри $a_i\ a_{i+1}$ образуват двуцифрено число (без водеща нула), което се дели на някое от числата d_1, d_2, d_3.

Данните се въвеждат от стандартния вход, където на един ред са записани последователно числата n, d_1, d_2, d_3, разделени с по един интервал.

Търсеният брой M да се изведе на стандартния изход.

Ограничения:

а) $1 < n < 20$ (за 50% от тестовите примери: $1 < n < 10$)

б) за всички тестови примери $M < 10000$.

Пример

Вход

```
3 13 28 34
```

Изход

```
15
```

Обяснение на примера:

134, 139, 265, 268, 284, 391, 526, 528, 565, 568, 652, 656, 684, 784, 913

Решение:

Следва пълният текст на програма, която решава задачата:

```
// Идея: Рекурсивно генериране на всички числа
#include <iostream>
using namespace std;
int n, d1, d2, d3;
int a[22];
int M=0;
bool OK(int a, int b)
// проверява дали две цифри a и b могат
// да бъдат една след друга в числото
// без водеща нула
{ if(a==0) return false;
  int x = 10*a + b;
  if(x%d1==0) return true;
  if(x%d2==0) return true;
  if(x%d3==0) return true;
  return false;
}
void Gen(int k) // избира k-тата цифра
{ for(int i=0; i<10; i++)
    if(OK(a[k-1],i))
    { a[k]=i;
      if(k<n) Gen(k+1); // рекурсивно търси (k+1)-вата цифра
      else M++;         // намерено е още едно от търсените числа
    }
}
int main()
{ cin >> n >> d1 >> d2 >> d3;
  for(int i=1; i<10; i++)
  { a[1]=i;  // първа цифра: 1,2,...,9
    Gen(2);  // генерира останалите цифри
  }
  cout << M << endl;
  return 0;
}
```

Задача D1. Календар

Училището в село Каръшко, което както е известно, признателното ръководство кръсти на наше име: "Програмистите на България", на 20.11.2005 г. ще празнува своя 70-ти юбилей. По случай празника директорът решил да организира състезание, като всеки един от участниците получил следната задача: по зададена дата (ден, месец, година) и какъв ден от седмицата е била тя, да се отпечата календарът за определен месец от същата година.

Умко (еднин от най-умните и мързеливи участници) ви моли да му помогнете да спечели състезанието, като напишете програма **CALENDAR**, която прочита от клавиатурата данните, зададени от директора и отпечатва календара на екрана на компютъра.

Програмата приема от първия ред на стандартния вход четири цели числа: деня, месеца, годината и деня от седмицата за известната дата (понеделник се отбелязва с 1, вторник – с 2, сряда – с 3 и т.н.), а от втория ред – само едно число – месеца, чийто календар трябва да се отпечата.

На стандартния изход, програмата отпечатва календара по следния начин: на първия ред се отпечатват първите букви на дните (на латиница) на седмицата, отделени с по два интервала. На следващите редове се отпечатват дните на зададения месец, започващи от 1, като всяка дата е под съответния ден от седмицата за този месец. Едноцифрените дни са разделени помежду си с по два интервала, а двуцифрените – с по един.

Примерен вход:

```
19 11 2005 6
1
```

Примерен изход:

```
P   V   S   C   P   S   N
                    1   2
3   4   5   6   7   8   9
10  11  12  13  14  15  16
17  18  19  20  21  22  23
24  25  26  27  28  29  30
31
```

Решение:

За да отпечатаме календара на даден месец трябва да определим какъв ден от седмицата е първият му ден. За целта трябва да се пресметне броят на дните между известната дата и първия ден на посочения месец. Тук трябва да обърнем внимание, че ако месецът е преди датата броя на дните се смята по един начин, а ако е след датата – по друг. Използва се масив, съдържащ броя на дните на всеки един от месеците, което улеснява пресмятането. Естествено, че трябва да съобразим да променим броя на дните на месец февруари за високосните години.

След като пресметнем броя на дните, чрез деление на 7 с остатък пресмятаме в кой ден от седмицата е първият ден от търсения месец. Когато получим това, остава само с помощта на един цикъл да отпечатаме и самия календар.

Ето една примерна програма, която решава горната задача:

```
#include <iostream.h>
void main()
{
  int d,m,g,d1,m1,sum=0,os,pom,i,j,br=0,
  a[13]={0,31,28,31,30,31,30,31,31,30,31,30,31};
   cin>>d>>m>>g>>d1>>m1;
   if(g%4==0)a[2]=29;
   if(m>=m1)
   {
```

```
    for(i=m-1;i>=m1;i--)sum+=a[i];
    sum+=d;
    os=sum%7;
     if(d1>os)pom=(d1-os)+1;
     else pom=(7-(os-d1)+1);
     pom=(pom%7)?pom%7:7;
    }
    else
    {
     for(i=m+1;i<m1;i++)sum+=a[i];
     sum+=(a[m]-d)+1;
     os=sum%7;
     pom=(os+d1)%7;
     if(pom==0)pom=7;
    }
  cout<<"P  V  S  C  P  S  N\n";
  for(i=1;i<=pom-1;i++)cout<<"   ";
  for(i=1;i<=(7-(pom-1));i++)cout<<i<<"  ";
  cout<<endl;
  for(j=i;j<=a[m1];j++)
   if(br!=6)
    {
      if(j>=10){cout<<j<<" ";br++;}
      else {cout<<j<<"  ";br++;}
    }
   else
    if(j>=10){cout<<j<<"\n";br=0;}
    else {cout<<j<<"  "<<"\n";br=0;}
   cout<<endl;
}
```

Задача D2. Милионер

Впечатлен от историята за забогатяване на Хенри Форд (започнал от една ябълка, продал я, купил 2 ябълки и т.н.), която госпожата в училището "Програмистите на България" разказала, Умко, който освен добър ученик е и футболен фен, си изградил следната стратегия за забогатяване, залагайки в „ Еврофутбол":

- За всеки нов тираж се прави един залог с коефициенти 2 или 3. Това значи, че ако заложим К лв. **печалбата** ще бъде съответно 2.К лв или 3.К лв.
- За всеки тираж заложената сума се определя от **печалбата** от предходен тираж увеличена с 1 лв. Всяка **печалба** се залага два пъти: веднъж с коефициент 2 и веднъж с коефициент 3. Спазва се правилото, че новата **печалба** не трябва да е по-малка от предходната.

Помогнете на Умко да намери след колко залагания, следвайки горната стратегия, ще стане милионер, ако в началото заложи 1 лв. Напишете програма **milioner.exe**, която въвежда цяло число ($1<n\leq 440000$) и извежда след най-малко колко залагания ще се получи **печалба** не по-малка от **n** и каква ще е стойността на тази **печалба**.

Вход: `9`
Изход: `5 9`
Печалбите са съответно: `2 3 6 8` **`9`**
Вход:25
Изход:11 26
Печалбите са съответно: `2 3 6 8 9 12 14 18 20 21` **`26`**

Коментар: Задачата се състои в генериране на елементи на множество по даден алгоритъм. Предложеното решение извършва тази генерация итерационно, като се използуват вече генерираните елементи съхранявани в масив и се избира по-малкия от новополучените. Използуват се два брояча за генерационните клонове за запазване на възходящия ред на получаване на елементите. С цел икономия на памет елементите неучастващи в следващите стъпки се изтриват. Стандартното решение ще покрие само част от тестовете.

```
#include <iostream.h>
void main()
{long y=0,i=1,i2=1,i3=1,k,n,x2,x3,
    x[15580] ;

 cin>>n;x[1]=1;
 do
 {   y++;x2=2*x[i2]+1;x3=3*x[i3]+1;
    i++;
    if (x2<=x3) { x[i]=x2;i2++; }
    else { x[i]=x3;
         for (k=2;k<=i;k++) x[k-1]=x[k];
         i--;i2--;
       }
  }
 while(x[i]-1<n);
cout<<y<<" "<<x[i]-1;
}
```

Задача D3. Думи в текст

За домашно по информатика учителката дала на Умко интересна задача, но той както обикновенно “няма време” да я реши и пак ще прибегне до вашата помощ. Все пак училището му носи вашето име: “Програмистите на България”. В задачата се иска да се напише програма **WORDS**, която прочита от първия ред на стандартния вход дума, съдържаща не повече от 20 знака, а от втория – произволен текст, съдържащ не повече от 1000 знака. Програмата трябва да отпечата номера на началния и крайния знак на най-дългата последователност от знаци в текста, които принадлежат на думата. Програмата трябва да може да различава малка от главна буква. Ако в текста не се среща нито една буква от думата, програмата извежда 0.

Примерен вход**:**
```
vaza
Programistite na Bulgaria se sastezavat.
```
Примерен изход**:**
```
35 38
```

Примерен вход**:**
```
ana
Atanas yade ananas
```
Примерен изход**:**
```
13 17
```

Примерен вход**:**
```
a
aaaa
```
Примерен изход:
```
1 4
```

Решение:

1. Необходими величини:

Две стрингови променливи, съответно за думата и текста.

```
char d[20], s[1000];
```

Пет целочислени променливи, съответно за дължината на думата и текста, началото и дължината на най-дългата последователност от знаци на думата, които се съдържат в текста, както и за началото и дължината на текущата последователност. Началната стойност на всяка една от тези променливи трябва да бъде 0.

```
int n, mn=0, md=0, tn=0, td=0, i, j;
```

Допълнително се декларират няколко работни променливи, които ще се използват за управление на циклите.

За определяне на принадлежността на даден знак към думата се използва помощен масив с 256 знака. Всеки елемент на този масив има стойност 0 ако знака със съответния код не е в думата и 1 – в противен случай. Този масив също се нулира в началото на програмата.

```
int b[256];
for(i=0; i<256; i++) b[i] = 0;
```

2. Въвежда се думата, намира се нейната дължина и се запълва масива b.

```
cin >> d;
n = strlen(d);
for(i=0; i<n; i++) b[d[i]] = 1;
```

3. Въвежда се текста и се определя дължината му:

```
cin.getline(s, 1000, '\n');
n = strlen(s);
```

4. Обхожда се текста и се намира най-дългата паследователност от от знаци на думата, които се съдържат в текста, като за всеки един от знаците на s, се проверява дали принадлежи на думата, т.е. дали съответното b е 1, при което се увеличава броя на буквите в текущата последователност от знаци, принадлежащи на думата. Ако знакът не принадлежи на думата са възможни два варианта: или това е края на текущата последователност и тогава трябва да проверим дали тя не е по-дълга от намерената до момента; или това е просто поредния знак от текста, който не принадлежи на думата и няма нужда от обработка.

```
for(i=0;i<n;i++)
     if(b[s[i]]){ if(!td)tn = i; td++;}
        else
          if(td)
            {
              if(td > md){md = td; mn = tn;};
              td=0;
            }
```

5. За да не се пропусне случая когато най-дългата последователност остава последна в текста, обработката за най-дълга последователност се повтаря след като цикъла завърши:

```
if(td > md){md = td; mn = tn;};
```

6. Накрая се извежда началото и края на намерената най-дълга последователност.

```
cout << mn + 1 << ' ' << mn + md << endl;
```

Задача E1. Числа

По време на дългите учебни часове нашият приятел Умко от училището в село Каръшко, което в наша чест е наречено "Програмистите на България", се забавлявал като измислял различни игри с числа. Случайно забелязал, че от едно петцифрено число, чрез разместване на цифрите му, могат да се получат много различни числа и дори ги преброил. Те били 120. Все пак написването им било досадна работа, дори когато ти е скучно в час. Умко решил, че не го интересуват всички тези числа, а само най-малкото и най-голямото от тях. Той започнал да разглежда различни петцифрени числа и за всяко от тях да намира разликата между най-голямото и най-малкото число, получени от неговите цифри. Скоро и тази работа му омръзнала, но все пак искал на всяка цена да знае тази разлика за всяко едно петцифрено число.

Сега вече само вие можете да му помогнете като напишете програма **NUMBERS.EXE**, която въвежда от клавиатурата на компютъра цяло петцифрено число N и извежда на екрана разликата между най-малкото и най-голямото измежду числата, получени като се разместят цифрите на N.

Примерен вход:
56342
Примерен изход:
41976

Обяснение:
Най-голямото число, което се получава от цифрите 2, 3, 4, 5 и 6 на числото 56342, е 65432, най-малкото е 23456, а разликата на тези две числа е точно 41976.

Примерен вход:
10002
Примерен изход:
19989

Обяснение:
Най-голямото число, което се получава от цифрите 1, 0, 0, 0 и 2 на числото 10002, е 20001, най-малкото е 12, а разликата на тези две числа е точно 19989.

Решение:
I начин: (Когато учениците не са учили масиви и оператори за цикъл.)
1. Необходими величини:

Три цели петцифрени променливи, които ще са от тип **long**, защото в тип **int** не се включват всички цели петцифрени числа, а само тези до 32767. В едната от тях ще се въведе числото N, а в другите две съответно минималното и максималното число, които се получават от цифрите на N.

```
long N, maxN, minN;
```

Освен тези три променливи ще ни трябва по една за всяка една от цифрите на числото N, които вече е добре да са от тип **int**.

```
int e, d, s, h, dh;
```

2. Въвежда се числото

```
cin >> N;
```

3. Пресмятат се цифрите му:

```
e=N%10;
d=N/10%10;
s=N/100%10;
h=N/1000%10;
dh=N/10000;
```

4. Подреждат се петте цифри по големина, като най-голяма става цифрата на десетохилядните, а най-малка цифрата на единиците:

```
if(e>d){int z=e;e=d;d=z;}
if(d>s){ z=s;s=d;d=z;}
if(s>h){ z=s;s=h;h=z;}
if(h>dh){ z=h;h=dh;dh=z;}
if(e>d){ z=e;e=d;d=z;}
if(d>s){ z=s;s=d;d=z;}
if(s>h){ z=s;s=h;h=z;}
if(e>d){ z=e;e=d;d=z;}
if(d>s){ z=s;s=d;d=z;}
if(e>d){ z=e;e=d;d=z;}
```

5. Получава се по-малкото от числата:

```
minN=e*10000+d*1000+s*100+h*10+dh;
```

6. Получава се по-голямото от числата:

```
maxN=dh*10000+h*1000+s*100+d*10+e;
```

7. Извежда се разликата между максималното и минималното число:

```
cout<<maxN-minN<<endl;
```

II начин: (Когато учениците са учили масиви и оператори за цикъл.)

1. Необходими величини:

Три цели петцифрени променливи, които ще са от тип **long**, защото в тип **int** не се включват всички цели петцифрени числа, а само тези до 32767. В едната от тях ще се въведе числото N, а в другите две съответно минималното и максималното число, които се получават от цифрите на N.

```
long N, maxN, minN;
```

Този път цифрите на числото записваме в масив от пет елемента:

```
int a[5], i, j;
```

2. Въвежда се числото

```
cin>>N;
```

3. Пресмятат се цифрите му:

```
for( i=0;i<5;i++){a[i]=N%10; N/=10; }
```

4. Подреждат се петте цифри по големина, като най-голяма става цифрата на десетохилядните, а най-малка цифрата на единиците:

```
for( i=0;i<4;i++)
   for(j=0;j<4;j++)
     if(a[j]>a[j+1]){ int z=a[j];a[j]=a[j+1];a[j+1]=z;}
```

5. Получават се двете числа:

```
maxN=minN=0;
for( i=0;i<5;i++)
  {
   minN=minN*10+a[i];
   maxN=maxN*10+a[4-i];
  }
```

6. Извежда се разликата между максималното и минималното число:

```
cout<<maxN-minN<<endl;
```

Ето окончателния вид на програмата от втория начин:

```
#include<iostream.h>
void main()
{
long N, maxN, minN;
int a[5], i, j;
cin >> N;
for( i=0;i<5;i++){a[i]=N%10; N/=10; }
for( i=0;i<4;i++)
  for(j=0;j<4;j++)
  if(a[j]>a[j+1]){ int z=a[j];a[j]=a[j+1];a[j+1]=z;}
maxN = minN = 0;
for( i=0;i<5;i++)
  {
  minN=minN*10+a[i];
  maxN=maxN*10+a[4-i];
  }
cout<<maxN-minN<<endl;
}
```

Задача Е2. Бонбонки

Умко (нашият познайник от училище "Програмистите на България") имал странно хоби. Той имал четири кристални купички, в които грижливо събирал шоколадови бонбонки. Странното било това, че той не обичал да си похапва от тях, а само да си ги гледа разпределени по равен брой в своите купички. Във всяка от тях той държал по 10 бонбонки. Това, че нашето умно момче не обичало шоколадовите бонбонки съвсем не означавало, че и неговата сестра не обичала да си похапва сладко, сладко от тях. Тя само изчаквала Умко да отиде на училище и с огромно удоволствие си похапвала. Така всеки ден след като се връщал от училище, той виждал, че любимите му кристални купички вече не са пълни с по 10 бонбонки във всяка и тичал до магазина, за да набави неоходимите бонбонки. След известно време Умко сменил тактиката. Вместо всеки ден да ходи до магазина и да си купува бонбонки, просто ги преместал от едната купичка в другата, за да станат пак по равен брои в четирите купички. В случаите, когато това не било възможно, Умко все пак тичал до магазина.

Тъй като му омръзнало всеки път да пресмята колко бонбони трябва да размести и да купи, нашият приятел ви моли да напишете програма **BONBONKI.EXE**, която прочита от клавиатурата последователно броя на бонбонките във всяка една от четирите купички и ако е възможно те да се разпределят по равно, извежда на екрана от коя купичка колко бонбонки се изваждат или добавят. Ако бонбонките се изваждат от купичката, пред броя им се поставя знака "–", а ако те се добавят – знака "+". Ако броя на бонбонките в съответната купичка остава непроменен, се извежда числото 0 без знак. Ако не е възможно бонбонките да се разпределят по равно в четирите купички, програмата извежда на екрана само броя на бонбонките, които Умко трябва да купи от магазина, за да поправи разпределението им.

Примерен вход:

```
2 3 4 6
```

Примерен изход:

```
1
```

Примерен вход:

```
4 9 8 7
```

Примерен изход:

```
+3 -2 -1 0
```

Решение:

За решението на задачата е необходимо да се намери общия брой бонбони във всички купички и да се намери остатъка при деление на 4 на този брой. Ако този остатък е 0, то резултатът при деление на общия брой на 4 е броя на бонбонките, който трябва да се получи във всяка купичка. Тогава за всяка купичка трябва да се провери дали трябва да се добавят и ли извадят бонбонки и да се отпечата съответния брой.

Ако остатъкът при деление на общия брой на 4 е различен от нула, се налага да се купят бонбонки от магазина и техният брой е точно допълнението на остатъка до 4.

Ето една примерна програма, която решава горната задача:

```
#include <iostream.h>
void main()
{
      int a,b,c,d,e;
      cin>>a>>b>>c>>d;
      e=(a+b+c+d)%4;
      if(e)cout<<(4-e)<<endl;
      else
      {
            e=(a+b+c+d)/4;
            if(a>e)cout<<"-"<<a-e<<" ";
            else if(a==e)cout<<"0"<<" ";
                        else cout<<"+"<<e-a<<" ";
            if(b>e)cout<<"-"<<b-e<<" ";
            else if(b==e)cout<<"0"<<" ";
                        else cout<<"+"<<e-b<<" ";
            if(c>e)cout<<"-"<<c-e<<" ";
            else if(c==e)cout<<"0"<<" ";
                        else cout<<"+"<<e-c<<" ";
            if(d>e)cout<<"-"<<d-e<<endl;
            else if(d==e)cout<<"0"<<endl;
                        else cout<<"+"<<e-d<<endl;
      }
}
```

Задача Е3. Познай цифрата

По време на междучасието в училището в село Каръшко играели следната игра: Подреждат се плочки, на всяка от които е изписана цифра от 0 до 9. Плочките са обърнати с надписа надолу, така че да не се вижда коя е цифрата на нея и се знае, че с тях са изписани числата от 10 до 99 в нарастващ ред (101112131415…979899).

Един от играчите посочва една от плочките, а този който е наред, познава коя цифра е записана на плочката. Печели този играч, който е познал най-много цифри. Нашият добър приятел Умко и този път иска да надхитри приятелите си, но за целта вие трябва да му помогнете като напишете програма **CIFRA.EXE**, която въвежда число k ($1 \le k \le 180$) и извежда цифрата изписана на k-тата плочка.

Пример:
Вход: 17
Изход 1

Вход: 28
Изход 3

Коментар на решението: Отчитайки, че числата са двуцифрени, определяме кое е числото, а от това дали е четно или нечетно ***k*** *се определя дали е цифрата на десетиците или на единиците. Задачата се решава само с един оператор* ***if.***

```
#include <iostream.h>
void main()
{
      int n,k,m;
      cin>>k;
      n=k / 20;
      m=k % 20;
      if (k%2)  cout<<n+1<<endl; else
      if (m) cout<<m/2-1<<endl;  else cout<<9<<endl;
}
```

Автори на задачите:
A1 – Веселин Райчев
A2 – Светослав Колев
A3 – Никола Борисов
B1 – Павлин Пеев
B2 – Емил Келеведжиев
B3 – Красимир Манев
C1 – Валентин Михов
C2 – Красимир Манев
C3 – Стоян Капралов
D1 – Петър Петров
D2 – Теодоси Теодосиев
D3 – Бисерка Йовчева
E1 – Бисерка Йовчева
E2 – Петър Петров
E3 – Теодоси Теодосиев

ЕСЕНЕН ТУРНИР ПО ИНФОРМАТИКА
Шумен, 10–12 ноември 2006 г.
Тема за група А (12 клас)

Задача А1. ИГРА

Даден е неориентиран граф с n върха и m ребра. Върховете са номерирани с числата от 1 до n. Дадени са и n пула, номерирани също с числата от 1 до n, като във всеки връх на графа е поставен по един пул. Разрешено е разместване на пуловете по един единствен начин: пул с номер 1 може да се размени с пул, който се намира в съседен връх на графа. Напишете програма **`move`**, която намира минималния брой ходове, за които пуловете могат да се разместят така, че всеки пул да се намира във връх със същия номер.

Вход: На първия ред на стандартния вход са дадени числата n и m. На следващите m реда са дадени по две числа, представляващи краищата на поредното ребро. На последния ред са дадени n числа, задаващи началното разположение на пуловете: i-тото число представлява номера на пула, намиращ се във връх i.

Изход: На стандартния изход да се изведе едно число – минималния брой ходове, с които може да се стигне до крайната позиция. Ако не е възможно да се достигне желаната позиция, да се изведе `-1`.

Ограничения: $n < 10$.

ПРИМЕР 1	**ПРИМЕР 2**
Вход	**Вход**

`3 2`
`1 3`
`2 3`
`3 1 2`
	`3 2`
	`1 2`
	`2 3`
	`3 1 2`

Изход	**Изход**
`2`	`-1`

Решение:

Да означим графа от условието на задачата с G. На всяко разположение на пуловете във върховете на G, съпоставяме пермутация $p = p_1, p_2, p_3, \ldots, p_n$, където p_i е номерът на пула, който се намира във връх i. Тъй като $n<10$, за представянето на различните пермутации ще използваме низове, например при n=5, низът `"23514"` означава, че във връх 1 се намира пул 2, във връх 2 – пул 3, във връх 3 – пул 5, във връх 4 – пул 1 и във връх 5 – пул 4.

Разглеждаме втори неориентиран граф P. Върховете на P са $n!$, като на всяка пермутация съответства връх от графа. Два върха са съседни, ако от едната пермутация може да се отиде в другата. Графът P е неориентиран, защото ходовете в играта са обратими. Например, ако в G върхове 2 и 4 са съседни, то от пермутацията `"23514"` може да се премине в `"21534"` и обратно.

Вместо да представяме графа P явно в програмата с някакви структури от данни, всеки път, когато трябва да намерим съседите на дадена пермутация-позиция извършваме следните пресмятания:

1) намираме върха u от графа G, в който се намира пул 1;
2) за всеки връх v, който е съсед на u (в графа G), получаваме съседна позиция в графа P, разменяйки пул 1 с пула от връх v.

При извършване на действията от точки 1) и 2), трябва да се съобразяваме с факта, че елементите на низовете са индексирани от 0.

В графа P трябва да намерим най-късия път (по брой ребра) от началната позиция до крайната позиция z="123…n". Ще използваме алгоритъма за търсене в ширина. За целта ни е необходима опашка, в която да бъдат включвани върховете (в нашия случай – позициите), които са достигнати за първи път.

Дефинираме и създаваме празна опашка по следния начин: `queue<string> Q;`

За всяка достигната позиция y, трябва да поддържаме броя ходове, с които може да бъде достигната. За целта ще използваме изображение (`map`), което на низове съпоставя числа, дефинирано така: `map<string,int> dist;`

Инициализираме стойностите на `dist` с `-1`, като за генерирането на всички пермутации използваме функцията от стандартната библиотека `next_permutation`.

Ако от текущата позиция x, може да се премине към съседна позиция y, за която `dist[y]` има стойност `-1`, то трябва да включим y в опашката: `Q.push(y)` и да установим `dist[y]=dist[x]+1`.

Броят на ребрата на графа P е равен на броя на ребрата на графа G. Следователно тялото на вътрешния цикъл в процедурата `bfs` се изпълнява общо m пъти. Едно отделно изпълнение зависи от реализацията на структурата `map`, която според спецификациите на стандартната библиотека осигурява ефективност $\log n!$. Така общо оценката за всички изпълнения на вътрешния цикъл е $O(m \log n!)$.

Всеки връх се посещава точно веднъж, което дава $O(n!)$ за операциите, свързани с опашката. Следователно за сложността на алгоритъма получаваме $O(n! + m \log n!)$ и като вземем пред вид, че $m \leq n(n-1)/2 \leq 36$, окончателно получаваме $O(n!)$ за сложността на алгоритъма по време. Очевидно и сложността по памет е $O(n!)$.

Стоян Капралов

Задача А2. СЛЪНЧОГЛЕДИ

Иванчо много обичал да ходи на гости на баба си Кунка. Тя имала къща с много дълъг балкон, на перваза на който били наредени саксии със слънчогледи. Противно на общоприетото схващане, те всъщност не гледали към слънцето. След като внимателно ги разгледал, Иванчо ограничил слънчогледите до четири типа – „северни“ (такива, които гледат на север), „южни“, „западни“ и „източни“. Установил още, че броят на северните слънчогледи е равен на броя на южните, както и, че броят на западните съвпада с броя на източните.

Баба Кунка размествала саксиите със слънчогледи всеки ден. При това не го правела безмозъчно, а с някаква странна логика и Иванчо скоро открил каква е тя: ако разгледаме всички саксии от някоя произволна позиция наляво, то броят на северните слънчогледи винаги е поне колкото на южните, както и броят на западните е поне колкото на източните. Изказано по-формално, горните правила звучат така:

n – брой саксии;
$SUM_N(i)$ – брой северни слънчогледи до саксия номер i ($1 <= i <= n$);
$SUM_S(i)$ – брой южни слънчогледи до саксия номер i ($1 <= i <= n$);
$SUM_W(i)$ – брой западни слънчогледи до саксия номер i ($1 <= i <= n$);
$SUM_E(i)$ – брой източни слънчогледи до саксия номер i ($1 <= i <= n$);

Изпълнено е, че:
$SUM_N(n) = SUM_S(n)$;
$SUM_W(n) = SUM_E(n)$;
$SUM_N(i) >= SUM_S(i)$ за всяко i, $i = 1, 2, ..., n$;

$SUM_W(i) >= SUM_E(i)$ за всяко i, $i = 1, 2, ..., n$.

Един ден Иванчо, гледайки поредната подредба на слънчогледите, се зачудил колко други подредби има, запазващи бабината логика. Той решил да счита, че саксиите с еднакво гледащи слънчогледи са неразличими, т.е. ако размени местата на две саксии с еднакво гледащи слънчогледи, няма да получи нова подредба. Помогнете на Иванчо – напишете програма **`flower`**, която прочита една конфигурация от слънчогледи и определя колко още други конфигурации съществуват.

Входните данни се състоят от два реда. На първия ред се намира четното естествено число n – броя саксии на балкона на баба Кунка. Този брой не надвишава 100. На следващия ред се намират n символа, разделени със по един интервал – това са типовете слънчогледи, разгледани от ляво надясно. С „**N**“ ще означаваме северен слънчоглед, с „**S**“ - южен, с „**W**“ - западен и с „**E**“ - източен. Изходът трябва да се състои от единствен ред с едно число – търсеният брой.

ПРИМЕР 1:	ПРИМЕР 2:
Вход:	**Вход:**
`6`	`8`
`N W E W S E`	`N N W E S S N S`
Изход:	**Изход:**
`29`	`139`

Решение:

Задачата може да се разглежда като просто обобщение на известната „задача за скобите“ (ако са ни дадени **n** двойки скоби, колко е броя на „правилните аритметични изрази“, които може да образуваме с тях). Тази задача има връзка с числата на Каталан, а именно, K_n изразява броя изрази с **n** двойки скоби.

Иначе казано, в дадената задача се пита по колко начина можем да образуваме израз с два различни типа скоби (например, кръгли и квадратни) – ако имаме **a** двойки кръгли и **b** двойки квадратни скоби, така че всеки тип скоби поотделно образуват правилен израз. Тъй като едните скоби не влияят на „правилността“ на другите, то имаме просто умножението на Каталановите им числа: $K_a * K_b$. Трябва, обаче, да вземем предвид как скобите са „размесени“. Тъй като целия израз има 2***a** + 2***b** символа, може просто да приемем, че започваме от 2(**a**+**b**) „празни“ клетки, в които трябва да попълним 2***a** полета със скоби от първия тип, а останалите полета запълним със другия тип. Това, очевидно, става по $C(2(\mathbf{a}+\mathbf{b}), 2\mathbf{a})$ начина.

Т.е. задачата може да се реши чрез следната формула:

$$A = K_a * K_b * C(2(\mathbf{a}+\mathbf{b}), 2\mathbf{a}) = C(2\mathbf{a}, \mathbf{a}) * C(2\mathbf{b}, \mathbf{b}) * C(2(\mathbf{a}+\mathbf{b}), 2\mathbf{a}) / ((\mathbf{a}+1)*(\mathbf{b}+1))$$

Където **a** е броят „северни“ слънчогледи (спрямо условието), а **b** е броят „западни“.

Решение 1 („Формулата“)

Това е първият начин да се реши задачата – достатъчно е да се намери формулата и да се смята по нея.

От формулата е достатъчно очевидно, че бройката расте много бързо, и дори при малки стойности на **a** и **b**, броят надхвърля обхвата на 32-битов тип. В тестовете има само 3 примера, които се „хващат“ с 32- (че дори и 64-) битова аритметика. Най-голямият отговор, който се получава при **a** = 25 и **b** = 25, е с *55* цифри. Явна е необходимостта от реализиране на „дълги

числа“ под някаква форма. Тук сложността откъм писане на задачата много зависи от пътя за реализация, които ще избере състезателя. Формулата горе, макар и проста, изисква тежките операции „умножение“ и „деление“, които са доста обемисти за писане. Лесно се вижда, обаче, че човек може да мине и без тях, защото множителите и делителите навсякъде са числа до 100. Казано на „програмистски жаргон“, необходими са само операции на умножение/деление на „дълги с къси“ числа.

Един по-ефикасен и удобен метод е през цялото време да пазим числата разложени в каноничен вид (т.е. всяко число X представяме във вида $X = a^x b^y c^z ...$). Така, например, за да умножим едно число X, което е вече в каноничен вид, с друго, Y, което не е, просто трябва разложим Y и да добавим простите множители и степените им в разлагането на X.

Пример:
$X = 90 = 2.3^2.5$, $Y = 14 = 2.7$
$X*Y = 2^2.3^2.5.7$

Така пресмятането на формулата горе се свежда до разлагане на множители и добавяне/изваждане в 100-елементов масив. Разбира се, накрая числото трябва да се превърне от каноничен вид в „нормален“. Тук отново се стига до „умножение на дълго с късо“ число, което не вярвам да затрудни състезателите.

Решение 2 (Динамично оптимиране)

Вторият начин за решаване на задачата се основава на техниката „динамично оптимиране“. За човек, който не е отлично запознат с числата на Каталан, биномните коефициенти и т.н. е по-логично да тръгне по този начин. Дефинира се четириаргументната функция f(N, S, W, E), с която означаваме броя начини да „довършим правилно“ израза, ако вече сме разположили N северни, S южни, W западни и E източни слънчогледа. Функцията много лесно се дефинира:

f(N, S, W, E) :=

0, ако N > **a** или S > **a** или W > **b** или E > **b** или S > N или E > W.
1, ако N = S = **a** и W = E = **b**.
f(N+1, S, W, E) + f(N, S+1, W, E) + f(N, S, W+1, E) + f(N, S, W, E+1), иначе.

Последния ред може да обясним със следното разсъждение: Ако вече сме разположили известен брой слънчогледи, то на текущата позиция можем да сложим някой от четирите типа слънчогледи и отговора е сумата от броя начини да довършим правилно израза при вече зафиксиран слънчоглед на текущата позиция.

Отговорът на задачата е стойността на f(0, 0, 0, 0). Директното прилагане на рекурсивната дефиниция по-горе, обаче, е непрактично, поради експоненциалната сложност на изчислението: всеки екземпляр от функцията (в общия случай) поражда 4 извиквания на същата функция. За целта ще използваме таблица dyn[51][51][51][51], в която ще записваме вече пресметнатите стойности на функцията (*memoization*).

Както вече споменахме, задачата изисква „дълги числа“ в някаква форма. Тук, обаче, за разлика от комбинаторното решение, нещата са по-приятни, тъй като се изисква само операцията „събиране“. Съществен проблем обаче започва да става обема на използваната памет: таблицата dyn е с 6,764,201 елемента и ако за всеки елемент отделим (например) 150 байта за едно дълго число, то заеманата памет ще стане към 1 GB. Има няколко начина да се разреши този проблем, между които:

- Да се определи каква е максималната големина на числата, които ще се съхраняват в таблицата. Това може да стане и експериментално. При една достатъчно пестелива реализация

на дълги числа, необходимата памет за таблицата става около 200 MB.

– Лесно се забелязва, че не всички клетки в таблицата ще се използват. Например, при **a** = 25, **b** = 25, таблица dyn[26][26][26][26] би ни свършила работа, а при **a** = 48, **b** = 2, дори по-малка таблица стига: dyn[49][49][3][3]. Това наблюдение може да се приложи по два начина: единият е да се задели толкова голяма таблица, колкото е нужно; вторият е, в таблицата да се съхраняват само указатели към числа, вместо самите числа, и паметта за всяка клетка да се заделя само когато е нужно.

Решение 3 (Прекалкулиране)

С използването на кой да е от двата начина горе, но без да се решават всички тънкости, е лесно да се изчислят отговорите за всеки възможен входен пример. Тъй като отговорите са симетрични за **a** и **b**, достатъчно е да се прекалкулират около 600 отговора.

Веселин Георгиев

Задача А3. ХРАНА

Гошо Ламерът е още ученик. Но вместо да ходи редовно на училище, той ходи редовно в близкия игрален клуб, където играе докато му свършат джобните. Един ден, когато това се случи, вече беше следобед и Гошо беше огладнял. Той отиде в близката верига за бързо хранене и застана пред вкусното меню с празни джобове. В заведението имаше много различни храни и напитки както и много промоции, с които ако човек си вземе например пица и кола спестява два лева. Може Гошо да е с празни джобове, но не е глупав. Той забеляза, че системата, която смята сметката първо прибавя цената на всички закупени храни и после изважда промоциите, за които има необходимите продукти. Гошо установи, че системата не проверява дали някой продукт вече е участвал в промоция и само с една кола например може да се хванат няколко промоции наведнъж (пица с кола, бургер с кола и т.н.)

Имайки всички продукти и промоциите за комбинации от тях, Гошо иска да му помогнете (като напишете програма **`food`**) да избере тези храни и напитки, за които не само няма да има нужда да плаща, но и да вземе максимално "ресто", за да се върне в игралния клуб.

Не забравяйте, че Гошо няма никакви пари, така че ако не може да поръча нещо без да плати, той няма да си купи нищо и ще се прибере вкъщи.

Входните данни се четат от стандартния вход. На първия ред има две числа N ($3 \leq N \leq 75$) и M ($1 \leq M \leq 75$), съответстващи на броя различни храни и броя промоции във веригата за бързо хранене. Храните и промоциите са номерирани от 1 до техния съответен брой. Следват N реда, съдържащи по едно цяло число C_i ($1 \leq C_i \leq 1000000$) – цената на съответната храна. Следват M реда започващи с числата P_i ($1 \leq P_i \leq 1000000$) и F_i ($1 \leq F_i \leq 10$) – отстъпката за съответната промоция както и броят храни, които участват в нея. Всеки от M-те реда завършва с F_i различни числа – от 1 до N – храните в съответната промоцията.

Вашата програма трябва да изведе на стандартния изход едно число – максимално ресто, което Гошо може да получи при подходящо подбрана поръчка. Ако няма храни, такива, че закупувайки ги не се налага да се плаща – изведете 0.

ПРИМЕР

ВХОД	**ИЗХОД**
5 4 20 20 20 100 10	10

```
10 1 1
60 3 1 2 3
20 2 3 4
10 1 5
```

Пояснение: Поръчвайки храни 1, 2, 3 и 5, Гошо ще хване промоции 1, 2 и 4, и ще е на печалба 10.

Решение:

Решението на задачата се свежда до построяване на претеглен насочен граф и намиране на максимален поток в него. Върховете на графа ще бъдат храните и промоциите, както и два специални върха – източник и цел. Ребрата в графа са както следва:

- Свързваме източника с всеки продукт, като теглото на насоченото ребро е цената на продукта.
- Свързваме всяка промоция с целта, като теглото на насоченото реброто е отстъпката на промоцията.
- Свързваме всеки продукт с промоциите, в които участва като за тегло на насочените ребра слагаме безкрайност.

В така построеният граф намираме максимален поток от върха източник до върха цел. Нека означим с $c[x][y]$ капацитета на ребро от връх x до връх y, а с $f[x][y]$ потокът преминал по реброто от връх x до връх y.

Печалбата, която Гошо може да постигне е равна на сумата на всички промоции минус големината на потока. Което всъщност е еквивалентно на сумата $(c[x][y] - f[x][y])$, за всички ребра излизащи от промоциите към върха цел.

За да достигнем до този извод може да разсъждаваме така – първоначално считаме, че всички промоции ни носят пари. Потокът, които минава през графа са парите които плащаме за храните. Той е с големина цените на храните. Също така, след като мине през някоя храна, потокът минава само през тези промоции, в които участва съответната храна – тоест той отнема от печалбата само на тези промоции. Следователно, щом за реброто на някоя промоция имаме $f[x][y] < c[x][y]$, то всички продукти за нея са изплатени (от нея и от други промоции) и ние ще добием печалба от нея.

Иван Анев

ЕСЕНЕН ТУРНИР ПО ИНФОРМАТИКА
Шумен, 10–12 ноември 2006 г.
Тема за група В (10-11 клас)

Задача В1. ФИБОНАЧИ

Разглеждаме редица от цели числа, в която първият и вторият член са равни на 1, а всеки следващ е равен на сумата на двата предишни за него члена в редицата. Напишете програма **`fib`**, която въвежда целите числа n и m, и извежда остатъка при делението с m на n-тия член на редицата ($0 < n < 10^{15}$, $0 < m < 10^7$).

Пример. Вход

```
1 1
```

Изход

```
0
```

Решение:

Наивният начин за пресмятане на n-тото число от редицата на Фибоначи ($f_1 = 1, f_2 = 2, f_n = f_{n-1} + f_{n-2}$) няма да може да реши задачата, поради твърде големия брой итерации за големи стойности на n:

```
long long int f1=1, f2=1, f;
for(long long int i=3;i<=n;i++)
 {
  f=f1%m+f2%m;
  f1=f2;
  f2=c%m;
 }
 cout << c;
```

Един възможен подход за решаване на задачата се основава на използване на формули, чрез които можем бързо да пресмятаме, например като скачаме от n-тия член f_n направо на $2n$-тия f_{2n}. Това може да се осъществи чрез двойката равенства, чието доказателство се извършва с математическа индукция:

$$f_{2n-1} = f_n^2 + f_{n-1}^2; \qquad f_{2n} = f_n^2 + 2 f_n f_{n-1}$$

или да използваме по-общото равенство:

$$f_{n+m} = f_{m+1} f_n + f_m f_{n-1}$$

(виж: Clifford A. Reiter. *Fast Fibonacci Numbers*, The Mathematica journal 2(3) 1992. Clifford A. Reiter, Fibonacci Numbers: Reduction Formulas and Short Periods, *The Fibonacci Quarterly*, 31(4) 1993, pp. 315-324. Воробиев, Н. Н., Числа Фибоначи, изд. Наука, Москва, 1978 г.)

Друг подход се основава на следната трансформацията, при която от дадена тройка последователни числа на Фибоначи, се получава следващата тройка: $(f_{i-2}, f_{i-1}, f_i) \rightarrow (f_{i-1}, f_i, f_{i+1})$ чрез матрично умножение:

$$\begin{pmatrix} f_{i+1} & f_i \\ f_i & f_{i-1} \end{pmatrix} = \begin{pmatrix} f_i & f_{i-1} \\ f_{i-1} & f_{i-2} \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

Ако означим

$$A_i = \begin{pmatrix} f_i & f_{i-1} \\ f_{i-1} & f_{i-2} \end{pmatrix}, \qquad f_0 = 0,\ \ i = 2, 3, ..., n,$$

пресмятането по модул m на степента $A_n = (A_2)^{n-1}$ може да се извърши с алгоритъм за бързо повдигане на степен чрез функцията `power`:

```
int a[][]={{1,0},{0,1}}, p[][]={{1,1},{1,0}};
void power(long long int n)
{
 if (n > 1) {power(n/2); a = mul(a,a);}
 if (n%2==1) a=mul(a,p);
}
```

където първоначално `a` е единичната матрица, `p` е заредена с елементите на A_2, а `mul` е функция за умножение на две матрици, като елементите на произведението се пресмятат по модул m. Това осигурява логаритмична изчислителна сложност вместо линейната сложност при наивния подход.

Емил Келеведжиев

Задача B2. ABC

Разглеждаме равностранен триъгълник с върхове точките A, B, C и с лице, равно на единица. Разделяме го на 4 по-малки еднакви равностранни триъгълника, като ползваме средите на страните му за върхове на новите триъгълници. Тези триъгълници означаваме съответно с буквите O, A, B и C, в зависимост от това, дали имаме предвид централния от тях, или тези, които са разположени откъм върховете A, B или C на първоначалния триъгълник. Всеки един от последните три по-малки триъгълници (триъгълниците A, B и C) го разделяме отново по същия начин на 4 още по-малки триъгълника. За да ги означим, ползваме също буквите O, A, B или C, но добавени към буквата, означаваща триъгълника, от който всеки нов триъгълник съответно е получен. Така имаме триъгълници AO, AA, AB, AC, BO, BA, BB, BC, CO, CA, CB и CC. Разглеждаме триъгълниците от това второ поколение, и за тези от тях, чието означение не завършва с O, повтаряме същата операция и получаваме следващото трето поколение триъгълници: AAO, AAA, AAB, ..., и т.н. Така продължаме построяването и на още следващи поколения триъгълници.

Напишете програма **abc**, която въвежда последователност от низове от описания вид (всеки низ задава триъгълник от някое поколение), и пресмята лицето на тази част от равнината, която е покрита със съответните на тези низове триъгълници. Лицето трябва да бъде изведено като обикновена несъкратима дроб, представена от числителя и знаменателя си, записани на един ред в стандартния изход и разделени с един интервал.

Вашата програмата трябва да прочете входните данни от стандартния вход. На първия ред във входа е зададен броят на следващите низове (цяло положително число, по-малко от 999). Следват толкова на брой редове, като всеки съдържа по един низ, състоящ се от някои от главни латински букви `A`, `B`, `C` или O. Дължината на всеки от тези низове не надминава 20 букви.

Пример. Вход

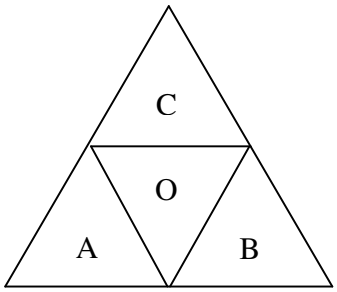
```
O
A
AB
```

Изход

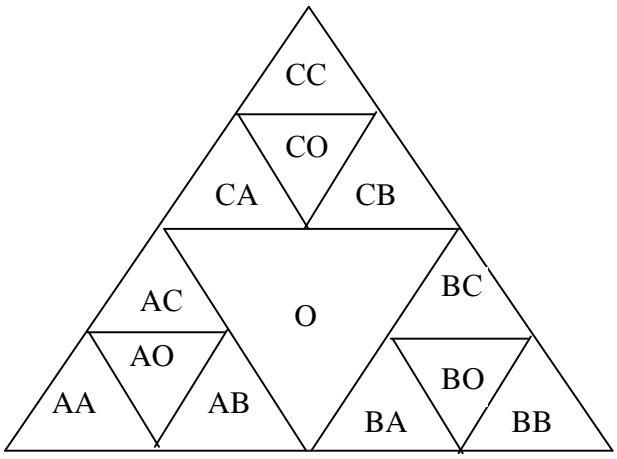
```
1 2
```

Решение:

При първото разделяне на триъгълници се получават четири триъгълника с лица равни на 1/4 от лицето на дадения триъгълник. Това са триъгълниците с имена A, B, C и O:



Второто поколение триъгълници имат лица, равни на 1/16 от лицето на първоначалния триъгълник:



За следващите поколения триъгълници, лицата са равни на 1/64, 1/256, и т.н., т.е. изразяват се като четна степен на 2.

За да решим задачата, трябва да премахнем тези от зададените във входните данни триъгълници, които са части от други триъгълници. За целта дадените низове се сортират лексикографски и след това преминавайки през така получената подредба, ако намерим, че за текущия низ, предходният му се съдържа от първа позиция в него, изключваме текущия. След това, в един числов масив запомняме дължините на непремахнатите низове. Всяка от тези дължини е равна на номера на поколението $k(t)$ на съответния триъгълник t. Понеже нашата цел е да съберем дроби от вида $1/(2^{k(t)})$, за намирането на общия знаменател търсим най-голямата от дължините $k(t)$. При пресмятането на степените на двойката може да използваме побитови измествания.

Накрая, ако е възможно съкращаваме дробта (като пробваме деления с 2 на числителя и знаменателя) и извеждаме резултата.

Зорница Дженкова

Задача В3. ПРАВОЪГЪЛНИ ТРИЪГЪЛНИЦИ

Правоъгълник с дължини на страните m и n (m и n са цели числа, за които $0 < m < 30$ и $0 < n < 30$) е разделен на mn квадрата със страна 1. Всяка точка, която е връх на поне един от тези квадрати, ще наричаме възел.

Да се напише програма **`rtri`**, която въвежда от един ред на стандартния вход стойности за m и n, и извежда на стандартния изход броя на правоъгълните триъгълници, върховете на които са възли.

ПРИМЕР

Вход
```
1 2
```

Изход
```
14
```

Решение:

Нека спрямо правоъгълна координатна система Oxy точките A, B и C имат координати $A(x_A, y_A)$, $B(x_B, y_B)$ и $C(x_C, y_C)$. Правите AB и AC са перпендикулярни тогава и само тогава, когато $(x_B - x_A)(x_C - x_A) + (y_B - y_A)(y_C - y_A) = 0$, а триъгълникът ABC е правоъгълен тогава и само тогава, когато $AB \perp AC$ или $AB \perp BC$ или $AC \perp BC$.

Да означим правоъгълника с $MNPQ$ и нека $MN = m$ и $MQ = n$. Нека спрямо правоъгълна координатна система координатите на върховете на правоъгълника са $M(0,0)$, $N(m,0)$, $P(m,n)$ и $Q(0,n)$. Абсцисата и ординатата на всеки възел могат да бъдат представени като елементи на два масива от цели числа $x[\ ]$ и $y[\ ]$, като на възел с координати (a,b) съпоставяме елементите $x[am+b] = a$ и $y[am+b] = b$.

Броят на всички правоъгълни триъгълници, върховете на които са възли, може да се намери като се изберат по всички възможни начини три възела и се провери дали избраните точки са върхове на правоъгълен триъгълник.

Младен Манев

ЕСЕНЕН ТУРНИР ПО ИНФОРМАТИКА
Шумен, 10–12 ноември 2006 г.
Тема за група С (8-9 клас)

Задача за С1. ПРОСТОТА ОТДЯСНО

Десетичният запис на естествените числа крие много любопитни свойства. Да наречем „просто отдясно" такова просто число, в десетичния запис на което:

- ако има повече от една цифра и изтрием цифрата на единиците – пак се получава просто число;
- ако полученото има повече от една цифра и направим същото – пак остава просто;
- и все така, докато остане само една цифра (и тя пак е просто число)!

Има ли „прости отдясно" числа? Да, вижте, например, числото 7193. Самото то е просто. Но прости са и всички числа, които се получават от него чрез описания по-горе процес – това са 719, 71 и 7. За да няма недоразумения, ще припомним, че числото 1 не е просто. За числото 5, например, също ще считаме, че е „просто отдясно" – то е просто и едноцифрено и за него описаният процес не започва (има нула стъпки).

Задачата Ви, която ще реализирате с програмата **`rprimes`**, се състои в преброяване на „простите отдясно" числа в зададен интервал [***a***, ***b***].

От стандартния вход се въвежда един ред с естествените числа ***a*** и ***b***, разделени с интервал, като $1 \le a \le b \le 2000000000$. Изведете на стандартния изход един ред с едно неотрицателно цяло число, равно на броя на простите числа с търсеното свойство, не по-малки от ***a*** и не по-големи от ***b***.

ПРИМЕР

Вход
```
7 300
```

Изход
```
13
```

Обяснение на изхода: В интервала [7, 300] „простите отдясно" числа са: 7, 23, 29, 31, 37, 53, 59, 71, 73, 79, 233, 239 и 293.

Решение:

Оказва се, че простите числа с описаното свойство са краен (и даже – малък) брой – всичко на всичко 83, като най-малкото от тях е 2, а най-голямото е 73939133. Генерирането им става лесно и бързо, което прави ненужно предварителното им изчисляване (но и това е допустимо на състезание): започвайки от списъка {2, 3, 5, 7} посочваме първото от тях и му долепваме отдясно всяка една от цифрите 1, 3, 7 и 9. Проверяваме всяко получено число за простота. Ако полученото е просто – добавяме го в края на списъка. Посочваме следващото в списъка и повтаряме процеса. Той приключва, когато показалецът надхвърли големината на текущия списък. Понеже числата се генерират наредени в нарастващ ред, броенето е лесно: прескачаме тези от списъка, които са по-малки от ***a*** и броим по-нататък в списъка тези, които са не по-големи от ***b***.

Примерна реализация на С:

```
#include <stdio.h>
```

```
#include <math.h>

unsigned long a,b,rprimes[1024]={2,3,5,7};
int count=4;

int isPrime(unsigned long a)
{unsigned long p,d;
 if (a==1) return 0;
 if (a==2) return 1;
 if (!(a&1)) return 0;
 d=ceil(sqrt(a));
 p=3;
 while (p<=d)
 {if (!(a%p)) return 0;
  p+=2;
 }
 return 1;
}

void makeRPrimes(void)
{int p=0;
 unsigned long d;
 do
 {d=10*rprimes[p]+1;
  if (isPrime(d)) rprimes[count++]=d;
  d+=2;
  if (isPrime(d)) rprimes[count++]=d;
  d+=4;
  if (isPrime(d)) rprimes[count++]=d;
  d+=2;
  if (isPrime(d)) rprimes[count++]=d;
  p++;
 } while (p<count);
}

int main (void)
{int c=0,i;
 scanf("%lu %lu",&a,&b);
 makeRPrimes();
 for (i=0;i<count&&rprimes[i]<a;i++);
 for (;i<count&&rprimes[i]<=b;i++,c++);
 printf("%d\n",c);
 return 0;
}
```

Примерна реализация на Pascal:

```
Var
 a,b:LongInt;
 rprimes: Array [0..1023] of LongInt;
 count,c,i:Integer;
```

```
Function isPrime(a:LongInt):Boolean;
Var p,d:LongInt;
Begin
 if a=1 then Begin isPrime:=FALSE; Exit; End;
 if a=2 then Begin isPrime:=TRUE; Exit; End;
 if Not odd(a) then Begin isPrime:=FALSE; Exit; End;
 d:=round(sqrt(a));
 p:=3;
 while p<=d do
 begin
  if a mod p=0 then Begin isPrime:=FALSE; Exit; End;
  p:=p+2;
 end;
 isPrime:=TRUE;
End;

procedure makeRPrimes;
Var p:Integer;
    d:LongInt;
Begin
 p:=0;
 Repeat
  d:=10*rprimes[p]+1;
  if isPrime(d) then Begin rprimes[count]:=d; Inc(count); End;
  d:=d+2;
  if isPrime(d) then Begin rprimes[count]:=d; Inc(count); End;
  d:=d+4;
  if isPrime(d) then Begin rprimes[count]:=d; Inc(count); End;
  d:=d+2;
  if isPrime(d) then Begin rprimes[count]:=d; Inc(count); End;
  Inc(p);
 Until p=count;
End;

BEGIN
 rprimes[0]:=2;rprimes[1]:=3;
 rprimes[2]:=5;rprimes[3]:=7;
 count:=4;
 ReadLn(a,b);
 makeRPrimes;
 c:=0;
 for i:=0 to count-1 do if rprimes[i]>=a Then break;
 for i:=i to count-1 do if rprimes[i]>b then break else Inc(c);
 WriteLn(c);
END.
```

Павлин Пеев

Задача C2. ПЛАТКА

Пешо, нашият прочут програмист, отишъл да учи в технически университет. Една от курсовите му задачи била да поправи една счупена платка. Платката представлявала точки, свързани с пътечки от алуминий, направени от поялник. За нещастие тези пътечки били разкъсани на някои места и за да я поправи трябвало да мине повторно с поялника върху тях. Платката работи само, ако всички точки са свързани с една непрекъсната пътечка от алуминий и заради това Пешо трябва да си избере една от точките, да сложи поялника на нея и без да го вдига да обходи всички останали точки. Проблемът бил, че ако мине два пъти по една и съща връзка между две точки – тя се поврежда. Помогнете на Пешо да се справи с тази трудна задача, като напишете програма **platka**, която по зададени връзките между точките, извежда реда, в който Пешо трябва да обходи точките.

Програмата приема входните данни от стандартния си вход. На първия ред са записани две числа N ($1 \leq N \leq 10000$) – броя на точките по платката и M ($1 \leq M \leq 100000000$) – броя на съществуващите пътечки между точките. Следващите M реда съдържат също по две числа – номерата на точките, свързани с пътечка.

Програмата трябва да изведе на стандартния изход на един ред начина на обхождане на точките, така, че след това платката да работи. Ако не съществува начин, при който да се обходят всички точки наведнъж, да се изведе съобщението “Sorry, Pesho”, следвано от броя на точките, който няма да бъдат достигнати, тръгвайки от избраната точка.

Пример:

Вход:
```
6 10
1 2
1 3
1 4
2 3
2 5
3 4
3 5
4 5
4 6
5 6
```

Изход:
```
1 3 5 6 4 1 2 3 4 5 2
```

Вход:
```
6 6
1 2
1 3
1 4
2 3
3 4
5 6
```

Изход:
```
Sorry, Pesho 2
```

Решение:

Решението на задачата прилага известния алгоритъм за намиране на Ойлеров път. (описанието на този алгоритъм може да се намери в различни книги за алгоритми). Единственото “усложнение” е изиксването за проверка дали графът има повече от една компонента:

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

int a[10000][10000];
int p[10000], vis[10000];
int n, m, u, v;

void read_data()
```

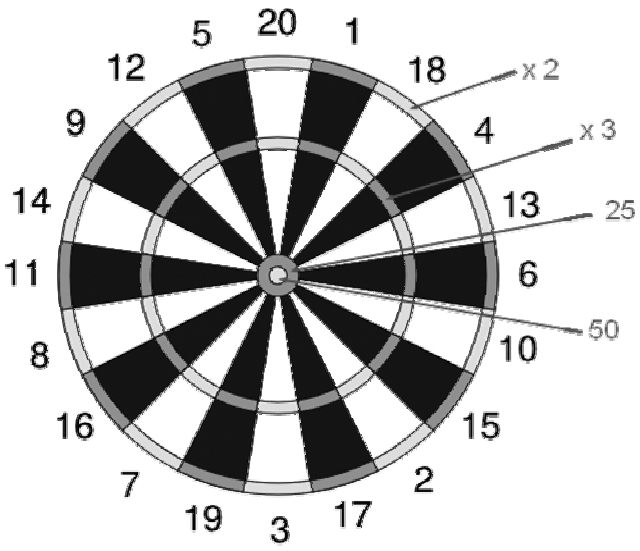
```
{
     cin >> n >> m;
     for (int i = 1; i <= m; i++)
     {
         cin >> u >> v;
         a[u][v] = 1;
         a[v][u] = 1;
         a[u][0] = !a[u][0]; //контролира се четността на върховете
         a[v][0] = !a[v][0]; //за четен връх – 0, а за нечетен – 1
     }
}

void dfs(int u) // за намиране броя на свързаните компоненти
{
     vis[u] = 1;
     for (int i = 1; i <= n; i++)
         if (a[u][i] && !vis[i])
            dfs(i);
}

void Euler()
{
     p[0] = u; //слага се първия връх в началото на пътя
     int l = 0, k = m; //l – края на началото, k – началото на края
     while (l < k) //докато не се срещнат
     {
           u = p[l]; //взема се последния връх от началото на пътя
           v = 1;
           while (v <= n && !a[u][v])
             //търси се има ли на къде да се ходи
                 v++; //цикъла спира или когато сме намерили
           if (v <= n) //съсед (v не е минало n)
           { //тогава
                 l++; //добавяме намерения съсед в началото на пътя
                 p[l] = v;
                 a[u][v] = 0; //премахваме връзките
                 a[v][u] = 0; //между двата върха
                 }
           else // ако сме обходили всички върхове
                 // и не сме намерили съсед
           {
               p[k] = u; //тогава този връх се маха от началото
               l--; k--; //и се слага в края на пътя
            }
      }
}

void print_data() // отпечатва пътя
{
     for (int i = 0; i <= m; i++)
         cout << p[i] << " ";
 }
```

```
int main()
{
    read_data();
    int i, br = 0;
    for (i = 1; i <= n; i++)
     if (a[i][0]) { u = i; break;} // намира се първия нечетен връх
   dfs(u);
        for (i = 1; i <= n; i++)
     if (!vis[i]) br++;
     //преброяват се всички необходени от DFS върхове
     if (!br) //ако няма такива
     {
          Euler(); //се намира ойлеровия път
          print_data(); // и се отпечатва
     }
     else
          cout << "Sorry, Pesho " << br << endl;
          //иначе се отпечатва броя
    return 0;
}
```

Антония Йовчева

Задача С3. DARTS

Дартс е игра, в която играчите хвърлят къси стрели в кръгла мишена, окачена на стената. Играта се е зародила преди няколко столетия на Британските острови. И досега дартс е традиционна игра във Великобритания, Нидерландия, Скандинавските страни, Съединените Щати и др.



Стандартната дъска за дартс е квадрат със страна 48 cm, центърът на който съвпада с центъра на мишената – окръжност с радиус 20 cm. Мишената е разделена на двадесет номерирани и еднакви по размер сектора, като сектор 20 е симетричен спрямо ординатната ос и секторите са номерирани по посока на часовниковата стрелка както следва: 20, 1, 18, 4, 13, 6, 10, 15, 2, 17, 3, 19, 7, 16, 8, 11, 14, 9, 12, 5.

В центъра на дъската са изобразени два концентрични кръга - с радиуси 0,5 cm и 1 cm, попадението в които се оценява съответно с 50 точки и 25 точки. Съществуват и два други концентрични обръча – вътрешен и външен. Вътрешният се определя от две окръжности с радиус

9,5 и 10,0 cm, а външният – от окръжности с радиуси 19,5 и 20,0 cm. Ако дадено попадение е между външния и вътрешния обръч или между вътрешния обръч и кръговете в центъра, се дават брой точки, равни на номера на сектора. Когато има попадение във външния обръч, играчът получава броят на точките за съответния сектор, умножени по 2. Ако попадението е във вътрешния обръч, точките от сектора се умножават по 3. Попадането на стреличка извън външния обръч не носи точки.

Когато дадено попадение е на границата на някой от гореописаните кръгове или обръчи, се счита, че попадението е в тази зона, която носи повече точки. Когато попадението е на границата между два сектора, се счита, че стреличката е попаднала в по-далечния сектор по посока, обратна на часовниковата стрелка (например, попадение между сектори 1 и 20 се счита в сектор 20, между 20 и 5 – в сектор 5, а между 17 и 2 – в сектор 2).

Във всеки рунд играчът хвърля 3 стрелички, като точките за рунда са сума от отделните попадения. Максимално възможният резултат е 180 точки (ако играчът попадне и с трите стрелички във вътрешния обръч на сектор 20).

Нека долният ляв ъгъл на дъската има координати (0,0), а горният десен – координати (48,48). Направете програма **darts**, която отчита броя точки за даден рунд при зададени координати на трите попадения.

Вход: От стандартния вход се въвеждат три двойки числа (x,y) – координатите на стреличките, с точност не повече от три знака след десетичната точка.

Изход: На стандартния изход се извежда единствено цяло число – броят точки за съответния рунд.

Примерен вход:	**Примерен изход:**
`2.212 1.423 24 24 24 34`	`110`

Решение:

Преди всичко, изчисляваме центъра на мишената с координати (24,24). За да се реши задачата, всъщност трябва да се открие отговорът на три подзадачи:

1. На какво разстояние от центъра на мишената е попадението? Това разстояние може да се сметне чрез Питагоровата теорема или разстоянието между точка A (x_1,y_1) и точка B (x_2,y_2) е $sqrt((x_1-x_2)^2+(y_1-y_2)^2)$.

Според разстоянието от центъра можем да направим следните изводи:

- При разстояние по-малко или равно на 0.5, се дават 50 точки.
- При разстояние по-малко или равно на 1.0, се дават 25 точки.
- При разстояние по-голямо от 20.0, се дават 0 точки.

В тези случаи попадението е отчетено и алгоритъмът приключва до тук.

В останалите случаи се определя коефициент за умножение M:

- При разстояние между 9.5 и 10.0, M=3.
- При разстояние между 19.5 и 20.0, M=2.
- Във всички останали случаи M=1.

2. Какъв е ъгълът между попадението и абсцисата на координатна система с център центъра на мишената? Ъгълът в градуси се пресмята по формулата:

$\angle\alpha = atan(y/x)*(180/pi);$

3. В кой сектор е попадението? Тъй като сектор 20 е симетричен спрямо ординатата, можем да се досетим, че обхваща интервала [81˚, 99˚). Нормализираме получения в подзадача 2 ъгъл: $\angle\alpha = \angle\alpha + 279^\circ$. Ако $\angle\alpha > 360^\circ$, то го нормализираме чрез $\angle\alpha = \angle\alpha - 360^\circ$. По този начин сектор 20 се намира в ъгловия диапазон от 0˚ до 18˚, сектор 5 – от 18˚ до 36˚ и т. н.

Всъщност, тъй като секторите са през 18 градуса, можем да използваме цялата част от делението $\angle\alpha / 18^{\circ}$. Секторът можем да получим чрез следната таблица:

Резултат от α/18	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Сектор	20	5	12	9	14	11	8	16	7	19	3	17	2	15	10	6	13	4	18	1

Накрая, получаваме брой точки, равен на M*Сектор.

Забележка: Откриването на ъгъла може да стане и без *atan*. Например, може да се използват отсечки, минаващи през центъра и чрез насочено лице да се сметне в кой от образуваните сектори попада дадена точка.

```
// Solution of "Darts" problem
#include <cstdlib>
#include <iostream>
#include <math.h>
#define pi 3.14159

using namespace std;

// Sector numbers

int Sectors[20]={
    20, 5, 12, 9, 14, 11, 8, 16, 7, 19,
    3, 17, 2, 15, 10, 6, 13, 4, 18, 1};

// Center of the target

double cx=24;
double cy=24;

double DistanceFromCenter(double x, double y)
// Returns the distance from the center
{
       return sqrt((x-cx)*(x-cy)+(y-cy)*(y-cy));
}

double Angle (double x, double y)
// Angle in degrees from the center of the target
{
       double ang=atan2(y,x)*(180/pi);
       if (ang<0) ang+=360;
       return ang;
       }

int GetSector (double x, double y)
// Returns the number of the sector
{
  double z=Angle(x-cx,y-cy)+279;
  while (z>360) z-=360;
```

```
  return (int) floor(z/18);
}

int Score(double x, double y)
// Calculates scores according to distance and sector
{
  double DfC=DistanceFromCenter(x,y);
  if (DfC<=0.5) return 50; else
     if (DfC<=1) return 25; else
          {
                 int MultiFactor=1;
                 if ((DfC>=19.5)&&(DfC<=20)) MultiFactor=2; else
                 if ((DfC>=9.5)&&(DfC<=10)) MultiFactor=3; else
                 if (DfC>20) MultiFactor=0;
                 int Sec=GetSector(x,y);
                 return Sectors[Sec]*MultiFactor;
          }
  cout<<'\n';
}
int main()
{   double x1,y1,x2,y2,x3,y3;
    cin>>x1>>y1>>x2>>y2>>x3>>y3;
    cout<<Score(x1,y1)+Score(x2,y2)+Score(x3,y3);
    cin>>x1;
}
```

Петър Събев

Шумен, 10–12 ноември 2006 г.
Тема за група D (6-7 клас)

Задача D1. Завещание

Един милиардер оставил на трите си дъщери - блондинки завещание, записано по странен начин. На лист с неразбираемо съдържание най-отдолу било написано:

P.S. В горния текст е закодирана сумата, която ще получите в наследство. За да няма сърдити, разделете парите поравно на трите.

За да помогнете на блондинките, напишете програма **MONEY.EXE**, която въвежда текст, завършващ с # , състоящ се от редове с не повече от 80 символа всеки, и извежда най-голямото число, открито в текста, което може да се раздели на трите поравно. Ако в текста няма такова число извежда “No”. Под число се разбира последователност от цифри заградена с други символи.

Примерен вход:

```
ffdgfdgfdg gdhb123gdhgh hgfhfj
hgkgh34
56gdfg dfgdfhg 71,90 fhjgfj ghjghkk-678gjghj gjkghk 4bhc
fgfg8nghjg
13 10
1hfgh gfhfghf ghjghj gkjghk jhlkhkl#
```

Примерен изход:

```
678
```

Примерен вход:

```
ffdgfdgfdg gdhb1239886gdhgh hgfhfj
hgfhfghf vfjghjhgk gjghkgk hjghjg2234567125690hkj jhkhjlkhjl
hjklhjlh
jj457990000000000000000000000000000000000000000000000000934
56gdfg dfgdfhg 78  90 fhjgfj ghjghkk678gjghj gjkghk 4bhc
ddssd gdfg
1234b
382569999999999999999999933333333333333333333331212121212   fgfg
gdfg
13
10
1
#
382569999999999999999999933333333333333333333331212121212
```

Примерен вход:

```
hgfhfghf vfjghjhgk gjghkgk hjghjg2hkj jhkhjlkhjl hjklhjlh jlkhjlh
hgkgh34
56gdfg dfgdfhg 71,97 fhjgfj ghjghkkgjghj gjkghk 4bhc
fgfg8nghjg ghmghkm#
```

Примерен изход:

No

Решение:

Решението изисква познаване на отделянето на елементи от текст, сравнение на многоцифрени числа и признак за делимост на 3. Програмата съдържа: функция за сравнение на многоцифрени числа; функция за отделяне на цифрите на число от текст и запомнянето им в масив с подравняване към определена дължина, едновременно с това се намира и сумата от цифрите на числото. Използвайки тези функции обхождаме целия текст отделяме число от текста, проверяваме дали е кратно на 3 и ако **да** го сравняваме с намереното до момента максимално число. Накрая се извежда най-голямото число кратно на 3.

```
#include<iostream.h>
#define d 80
int a[d],b[d],n,m;
char ch;
void sr(int x[],int y[],int z[])
{int k,i=0;
 while (i<d&&x[i]==y[i]){z[i]=x[i];i++;}
 if (x[i]>y[i])for(k=i;k<d;k++) z[k]=x[k];
 else for(k=i;k<d;k++) z[k]=y[k];
 //for (k=0;k<d;k++)cout<<z[k];
}

int st(int x[])
{int k,i=-1,br=0;
  while (ch>='0'&&ch<='9')
   {i++;
    x[i]=ch-'0';
    br+=x[i]; cin.get(ch);
```

```
   }
 for (k=i;k>=0;k--) x[d-1-i+k]=x[k];
 for (k=0;k<d-1-i;k++) x[k]=0;
return br;}

void main()
{
 a[0]=-1;for(m=1;m<d;m++)  a[m]=0;
 do
 {cin.get(ch);
 if (ch>='0'&&ch<='9')  if (st(b)%3==0)  sr(a,b,a) ;
 }
 while (ch!='#');
 if (a[0]==-1) cout<<"No";
  else { n=0; while (n<d&&a[n]==0) n++;
 if (n==d) cout<<0; else for (m=n;m<d;m++)cout<<a[m];
       }
 cout<<"\n";
 }
```

Решение на задачата на Pascal

```
program aa;
const d=80;
type mas=array[1..d]of integer;
var a,b:mas;n,m,i:integer;ch:char;
procedure sr(x,y:mas;var z:mas);
var i:byte;
begin i:=1;
 while (i<=d) and (x[i]=y[i]) do
  i:=i+1;
 if x[i]>y[i] then z:=x
 else z:=y;
end;

procedure st(var x:mas;var br:integer);
 var k,i:integer;
 begin
  i:=0;br:=0;
  while ch in ['0'..'9'] do
   begin
    i:=i+1;
    x[i]:=ord(ch)-ord('0');
    br:=br+x[i]; read(ch);
   end;
 for k:=i downto 1  do x[d-i+k]:=x[k];
```

```
 for k:=1 to d-i do x[k]:=0;
 end;

begin
 a[1]:=-1;for m:=2 to d do a[m]:=0;
 repeat
 read(ch);
 if ch in ['0'..'9'] then begin
                          st(b,i);
                          if i mod 3=0 then sr(a,b,a)
                          end
 until ch='#';
 if (a[1]=-1) then write('No') else
 begin
 n:=1; while (n<=d)and(a[n]=0) do n:=n+1;
 if n>d then write(0) else for m:=n to d do write(a[m]);
 end;
 writeln;
end.
```

Теодоси Теодосиев

Задача D2. Влакове

В село Клавиатурмишково са открити находища на нефт и въглища в толкова големи количества, че за да се извозят е нужно да се създаде отделна ж.п. гара.

Г-н Мишо Клавишов, кметът на село Клавиатурмишково е изправен пред сериозен проблем! Нужна му е компютърна програма TRAINS.EXE, която чертае схема на гарата, заедно с влаковете, които в даден момент са спрели на гарата.

На ж.п. гарата има N (1≤N≤9) коловоза, всеки от които дълъг по M (1≤M≤50) единици, като единица ж.п. линия на схемата се отбелязва със символа „#”. Всеки от влаковете има точно по един локомотив, който на схемата се отбелязва с “(H H]”. Всеки влак може да има два вида вагони – за въглища (отбелязва се с „[XXX]”) и вагон-цистерна (отбелязва се с “(O O)”). Вагоните с въглища се композират винаги преди вагоните-цистерни. Броят на вагоните е така зададен, че целият влак да може да се визуализира на схемата.

Входни данни

На първия ред от клавиатурата се въвеждат 2 цели числа – N и M. На всеки от следващите N реда има по три цели числа – P, C, F.

P е позицията на локомотива в брой единици от началото на коловоза. Ако P=0, то коловозът е празен и стойностите на C и F за този коловоз нямат значение.

C е броят на вагоните с въглища.

F е броят на вагоните-цистерни.

Изходни данни

На екрана се извежда схема на гарата, заедно с влаковете, като всеки коловоз се показва на нов ред.

Пример:

В примера по-долу правим схема на гара с 3 коловоза, всеки от които с дължина 32 единици. На 1-ви коловоз имаме влак, който започва от 4-та позиция и има 1 вагон с въглища и 2

вагона-цистерни. 2-ри коловоз е празен. На 3-ти коловоз има влак, който започва от 3-та позиция и има 5 вагона-цистерни.

Примерен вход:

```
3 32
4 1 2
0 2 3
3 0 5
```

Примерен изход:

```
###(H H][XXX](O O)(O O)#########
################################
##(H H](O O)(O O)(O O)(O O)(O O)
```

Решение:

За решението на задачата ще използваме три едномерни масива, в които ще записваме съответно – в pos[i]- позицията на локомотива в i-тия коловоз, в c[i] – броя на вагоните с въглища, а в f[i]- броя на цистерните.

Единствената важна преценка е да се изчисли броят на символите, оставащи след последния вагон.

Те се изчисляват, като от дължината на коловоза M се извади:

5 * (1 + F + C) + позиция на локомотива, където F е броя на вагоните с въглища, а C- броя на цистерните.

Реализация на алгоритъма на езика C++

```
#include <cstdlib>
#include <iostream>
using namespace std;
int main()
{
int i,j,m,n;
int pos[9],c[9],f[9];
cin>>n>>m;
for (i=0;i<n;i++)
  cin>>pos[i]>>c[i]>>f[i];
for (i=0;i<n;i++)
  {
  if (pos[i]>0)
    {
      for (j=1;j<pos[i];j++) cout<<'#';
      cout<<"(H H]";
      for (j=0;j<c[i];j++) cout<<"[XXX]";
      for (j=0;j<f[i];j++) cout<<"(O O)";
      for (j=5*(1+c[i]+f[i])+pos[i];j<=m;j++) cout<<'#';
    }
  else
    for (j=0;j<m;j++) cout<<'#';
  cout<<'\n';
  }
```

```
}
```

Петър Събев

Задача D3. Библиотека

Първата задача на библиотекарката Книжка Книгова на новото и работно място не била от най – любимите и. Тя трябвало да подреди N книги на шкаф с много рафтове, така че след като някой поиска дадена книга да може да я намира за най – малко време. Ако подреди всичките N книги само на един рафт, времето необходимо за намирането, на коя да е книга ще е N секунди, а ако раздели книгите на K рафта, то времето, необходимо за намирането на книга от кой да е рафт ще е K + P секунди, където P е броя на книгите на съответния рафт (т.е. K секунди, за да намери рафта и още P секунди, за да намери търсената книга). Например, ако има 7 книги, Книжка може да сложи всички книги на един рафт и тогава ще и отнеме 7 секунди за да намери коя да е от тях. Ако тя раздели 7 книги на 2 рафта с по 3 и 4 книги всеки, то ще и отнеме първо 2 секунди за да намери правилния рафт и след това още 3 или 4 секунди за намиране на всяка една от книгите, но след като незнаем коя книга ще търсим общото време за търсене ще бъде 6 секунди. Помогнете на библиотекарката да разбере колко ще е минималното време, необходимо намиране на произволна книга.

Напишете програма **MINITIME.EXE**, която от стандартния вход чете едно цяло число N(1<=N<=107) – броя на книгите, които Книжка трябвало да подреди и отпечатва на стандартния изход едно число – минималното време, за което тя ще може да намери коя да е от книгите.

Примерен вход:
7
Примерен изход:
6

Примерен вход:
3
Примерен изход:
3

Решение:

Очевидно е, че трябва да намираме минималното време разпределяйки равномерно книките по рафтове, започвайки от разпределение на книгите на един рафт, след това на два рафта и така разпределяйки ги до N/2 рафта включително. Тук интересно е смятането на времето за всяко едно разпределение по рафтове. Ако разделим N книги на K рафта то имаме две възможности. Първата е ако книгите се разпределят по равно на вскичките K рафта т.е. K дели N без остатък. Тогава минималното време ще е K секунди (броя на рафтовете) и още N/K секунди – броя на книгите на един рафт. Но ако книгите не могат да се разделят по равно на вскичките K рафта, тогава трябва да ги разпределим на K+1 рафта, където първите K рафта ще са с по равен брой книги, а K+1-вия ще е с толкова книги колкото са останали след разполагането на първите K рафта т.е. с N%K книги. Тогава минималното времето ще е K+1 секунди (за боря на рафтовете) и още N/K секунди за броя на книгите, защото на K+1-вия рафт броя на книгите ще е винаги по – малък от колкото в горните рафтове.

Реализация на алгоритъма на езика C++

```
#include <iostream>
using  namespace std;
int main()
```

```
{
  int n,mint,i,p,t;
  cin>>n;
  mint=n;
  for(i=2;i<=n/2;i++)
  {
       if(n%i)p=1;
       else p=0;
       t=(n/i)+(i+p);
      if(t<mint)mint=t;
  }
  cout<<mint<<endl;
  return 0;
 }
```

Петър Петров

ЕСЕНЕН ТУРНИР ПО ИНФОРМАТИКА

Шумен, 10–12 ноември 2006 г.

Тема за група Е (4-5 клас)

Задача Е1. Треска за злато

Седем участници се състезават на „златната пирамида" в „Треска за злато". Всеки участник прави три опита. Ако поне при два от опитите участник улучи един и същ резултат, той печели сумата от трите опита. Кой от участниците ще спечели най-голяма печалба?!

Напишете програма **GOLD.EXE**, която въвежда седем реда с по три числа (по-малки от 100000). Числата на всеки ред са резултатите от трите опита на поредния състезател.

Програмата извежда номера на победителя и спечелената сума. Ако никой не улучи два пъти един и същ резултат, програмата трябва да изведе "No". Ако двама или повече състезатели наберат еднаква сума, печели първият постигнал тази сума.

Примерен вход:
300 4444 300
224 80000 7780
5500 5500 4100
8978 9000 6789
6000 6000 3100
6000 4550 4550
2222 8980 900
Примерен изход:
3 15100

Примерен вход:
5300 4440 3000
224 80000 7780
5000 5500 4100
8978 9000 6700
6000 600 3100
6000 4550 47500
2200 8980 900
Примерен изход:
No

Коментар на решението:

Решението на задачата изисква познаване само на оператор **if** и съставен оператор (блок). Проверяват се възможностите за използване на съставни условия и търсене на максимален елемент. Разбира се, че стилното решение е с организиране на цикъл.

Рещение на задачата на езика Pascal:

```
program gold;
var a,b,c,p,max:longint;br,nom:byte;
 begin
 readln(a,b,c);max:=0;br:=1;
 if (a=b) or (a=c) or (b=c) then begin
                                    p:=a+b+c;
                                    if      p>max      then      begin
max:=p;nom:=br end
                                    end;
 readln(a,b,c); br:=br+1;
 if (a=b) or (a=c) or (b=c) then begin
                                    p:=a+b+c;
                                    if      p>max      then      begin
max:=p;nom:=br end
                                    end;
 readln(a,b,c); br:=br+1;
 if (a=b) or (a=c) or (b=c) then begin
                                    p:=a+b+c;
                                    if      p>max      then      begin
max:=p;nom:=br end
                                    end;
 readln(a,b,c);  br:=br+1;
 if (a=b) or (a=c) or (b=c) then begin
                                    p:=a+b+c;
                                    if      p>max      then      begin
max:=p;nom:=br end
                                    end;
 readln(a,b,c); br:=br+1;
 if (a=b) or (a=c) or (b=c) then begin
                                    p:=a+b+c;
                                    if      p>max      then      begin
max:=p;nom:=br end
                                    end;
 readln(a,b,c); br:=br+1;
 if (a=b) or (a=c) or (b=c) then begin
                                    p:=a+b+c;
                                    if      p>max      then      begin
max:=p;nom:=br end
                                    end;
 readln(a,b,c); br:=br+1;
 if (a=b) or (a=c) or (b=c) then begin
                                    p:=a+b+c;
                                    if      p>max      then      begin
max:=p;nom:=br end
                                    end;
```

```
 if max>0 then writeln(nom,'  ',max)
 else writeln('No');
 end.
```

Решение на задачата за тези, които са учили цикли:

```
program gold;
var a,b,c,p,max:longint;i,br,nom:byte;
 begin
 max:=0;br:=0;
 for i:=1 to 7 do
 begin
 readln(a,b,c); br:=br+1;
 if (a=b) or (a=c) or (b=c) then begin
                                  p:=a+b+c;
                                  if     p>max     then     begin
max:=p;nom:=br end
                                  end;
 end;
 if max>0 then writeln(nom,'  ',max)
 else writeln('No');
 end.
```

Решение на задачата на езика C++:

```
#include<iostream.h>

void main ()

{long a,b,c,p,max=0;

 int i,br=0,nom;

 for (i=1;i<=7;i++)

 {

 cin>>a>>b>>c; br++;

 if (a==b||a==c||b==c) { p=a+b+c;

               if (p>max) { max=p;nom=br;}

             }

 }

 if (max) cout<<nom<<" "<<max;

 else cout<<"No";
```

```
}
```

Теодоси Теодосиев

Задача Е2. Дата

Младият програмист Росенчо се готвил за есенния турнир по информатика в гр. Шумен и бил освободен от училище. Един ден негов приятел му звъннал по телефона и му съобщил, че една седмица след завръщането му от турнира ще имат контролна. Росенчо започнал да пресмята на коя дата точно е контролната му и съвсем се объркал.

Помогнете му като съставите програма **DATA.EXE**, която въвежда от стандартния вход дата включваща ден, месец и година, разделени с по един интервал.

На един ред на стандартния изход трябва да се изведе датата след седем дни, като деня, месеца и годината са разделени с тирета (-).

Не забравяйте, че една година е високосна, ако се дели на 4, но не се дели на 100 или ако се дели на 400!

Примерен вход:	**Примерен вход:**	**Примерен вход:**
`12 11 2006`	`28 2 2004`	`31 12 2006`
Примерен изход:	**Примерен изход:**	**Примерен изход:**
`19-11-2006`	`6-3-2004`	`7-1-2007`

Решение:

Нека входът се записва в три променливи: `d` – ден, `m` – месец, `g` – година. Прибавяме към `d` 7 и ако получената стойност е по-голяма от дните в месеца (`m`), от променливата `d` изваждаме дните на въведения месец и прибавяме единица към месеца (`m`). Ако месецът е по-голям от `12`, то следващият месец трябва да е януари (т.е. на променливата m присвояваме `1`) и годината увеличаваме с единица. Когато въведената дата е през февруари трябва да се прави проверка дали годината е високосна. Както знаем, една година `g` е високосна, когато се дели на `4` без остатък, като специално правило се прилага, когато последните две цифри на `g` са нули. Тогава допълнителното условие годината `g` да е високосна е да се дели на `400` без остатък. Това се изразява чрез следното съставно логическо условие:

```
(((g%4==0)&&(g%100!=0))||(g%400==0))
```

Намерените `d`, `m`, `g` извеждаме разделени с тире.

Ето една примерна програма, която решава горната задача:

```
#include <iostream>
using namespace std;
int main()
{
        int d,m,g;
        cin>>d>>m>>g;
```

```
        d=d+7;
        if ((m==1)||(m==3)||(m==5)||(m==7)
         ||(m==8)||(m==10)||(m==12))
        if (d>31)
              {
                  d=d-31;
                  m=m+1;
               }
        if ((m==4)||(m==6)||(m==9)||(m==11))
        if (d>30)
               {
                 d=d-30;
                 m=m+1;
               }
       if (m==2)
       if ((((g%4)==0)&&((g%100)!=0)) ||((g%400)==0))
        {
        if (d>29)
              {
                  d=d-29;
                  m=m+1;
               }
        }
       else {
        if (d>28)
                {
                  d=d-28;
                  m=m+1;
                }
           }

     if (m>12)
            {
                  m=1;
                  g=g+1;
           }
     cout<<d<<"-"<<m<<"-"<<g<<endl;
     return 0;
}
```

Бистра Танева

Задача Е3. Текст

При подготовката на есенния турнир се обменят много съобщения. Специалната пощенска кутия на турнира била нападната от вируси и заглавията на писмата в нея се разбъркали. За щастие писмата били запазени, но било трудно да се определи по заглавията кои от тях се отнасят за турнира и кои са случайно попаднали (спам). Известно е, че писмата, свързани с турнира имат трибуквени заглавия и задължително съдържат поне една от буквите на съкращението на турнира NET – Национален Есенен Турнир. За да помогнете на организаторите да открият интересуващите ги писма напишете програма **NET.CPP**, която прочита от

клавиатурата три букви и проверява колко от тях се срещат в съкращението NET. Буквите могат да бъдат малки и главни, например “n” и “N” са една и съща буква.

Примерен вход:
KeT
Примерен изход:
2
Примерен вход:
nnn
Примерен изход:
3
Примерен вход:
IOI
Примерен изход:
0

Решение:
Задачата изисква познаване на знаков тип **char**.
Небходими величини:
Една знакова порменлива, с която ще се четат от клавиатурата последователно трите знака.
char c;
Edna целочислена променлива, чиято първоначална стойност ще бъде 0, за броя на буквите, които се съдържат в думата NET.
int br=0;
1. Въвежда се първия знак:

cin>>c;
2. Проверява се дали този знак се среща в думата NET, като се вклу4ва и проверката за главни и малки букви. Ако знака се среща в думата, променливата br(брояча) се увеличава с 1.

if(c=='N'||c=='n'||c=='E'||c=='e'||c=='T'||c=='t')br++;
3. Повтаря се същото действие и за другите две числа.
4. Накрая се извежда брояча.

Ето една примерна програма, която решава горната задача:

```
#include <iostream>
using namespace std;
int main()
{
    char c; int br;
    cin>>c;
    if(c=='N'||c=='n'||c=='E'||c=='e'||c=='T'||c=='t')br++;
    cin>>c;
    if(c=='N'||c=='n'||c=='E'||c=='e'||c=='T'||c=='t')br++;
     cin>>c;
    if(c=='N'||c=='n'||c=='E'||c=='e'||c=='T'||c=='t')br++;
    cout<<br<<endl;
}
```

Бисерка Йовчева