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 $len \le lenr$. 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 ($len \le lenr$), 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, $2 \le K \le 36$. 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 ($1 \le R \le 1000000$) (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 $R \le 6$.

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

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.

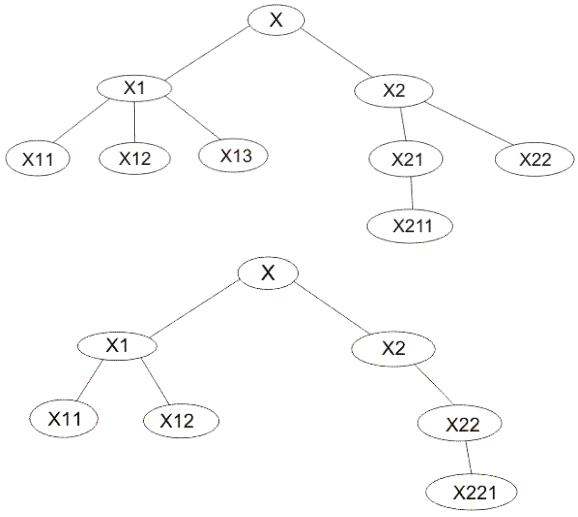
Входните данни се четат от стандартния вход. Дървовидната структура на папките е зададена, започвайки с главната папка, чието име е написано в първия ред на входния файл (низ от малки латински букви и цифри), слевано от броя на намиращите се в нея подпапки и/или файлове и времето на създаването й (в 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.

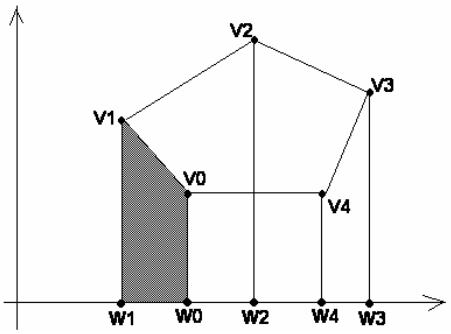
Входните данни ще бъдат зададени на стандартния вход. На всеки ред ще бъде зададено описанието на един от многоъгълниците. То започва с броя M на върховете на многоъгълника ($3 \le M \le 100$), последван от M-те двойки целочислени координати на върховете (в правоъгълна координатна система), зададени 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.

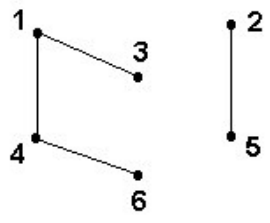
Пример.
Вход:
```
6 4
1 3
1 4
2 5
4 6
```

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