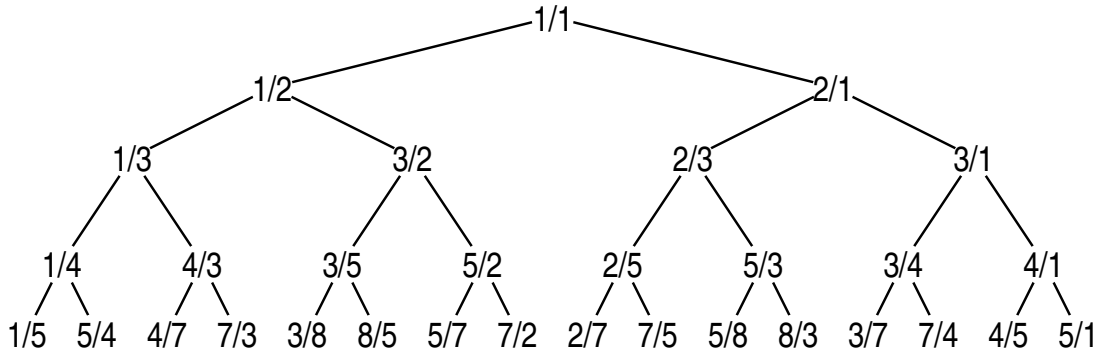
ЗИМНИ МАТЕМАТИЧЕСКИ СЪСТЕЗАНИЯ

Русе, 1-3 февруари 2008 г.

Група А, 11-12 клас

Задача А1 • РАЦИОНАЛНИ ЧИСЛА

Да построим едно безкрайно двоично дърво с върхове, надписани с положителни рационални числа по следния начин: коренът на дървото е надписан с числото $\frac{1}{1}$ и за всеки връх на дървото, надписан с числото $\frac{r}{s}$, левият син на върха е надписан с числото $\frac{r}{r+s}$, а десният син – с числото $\frac{r+s}{s}$. За по-голяма яснота предлагаме картинка:



Нека $f(0)$, $f(1)$, ..., $f(n)$, ... е изброяване на върховете на дървото, получено чрез обхождане в ширина. Така $f(0) = \frac{1}{1}$, $f(1) = \frac{1}{2}$, $f(2) = \frac{2}{1}$, $f(3) = \frac{1}{3}$, $f(4) = \frac{3}{2}$, ... Напишете програма `RAT`, която по дадено число n намира рационалното число $f(n)$.

Вход
От единствен ред на стандартния вход се въвежда числото n.

Изход
Програмата извежда на един ред числителя и знаменателя на рационалното число $f(n)$, разделени с точно един интервал.

Ограничения:
$0 \leq n \leq 2\,000\,000\,000$

Пример 1

Вход	Изход
`0`	`1 1`

Пример 2

Вход	Изход
`5`	`2 3`

Пример 3

Вход	Изход
`10`	`5 2`

РЕШЕНИЕ

Всъщност може да се види, че в дървото има само несъкратими дроби и всяка несъкратима дроб с положителен числител и знаменател се среща точно веднъж в дървото.

Първото свойство, което се вижда е, че знаменателят на $f(n)$ е равен на числителя на $f(n+1)$ за всяко $n \geq 0$. След това да образуваме редицата от естествени числа $b(0), b(1), \ldots, b(n), \ldots$ по следния начин: $b(n)$ е числителят на $f(n)$. От първото свойство получаваме, че $f(n) = \dfrac{b(n)}{b(n+1)}$. Сега за редицата $b(n)$ имаме рекурентните връзки $b(2n+1) = b(n)$ и $b(2n+2) = b(n) + b(n+1)$.

Тези връзки дават решение на задачата със сложност $O(\log^2 n)$.

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

long b(long n)
 {
  if (n==0) return 1;
  if (n % 2 == 1) return b(n/2);
  return b(n/2) + b(n/2 - 1);
 }

int main()
{ long n;
  cin >> n;
  cout << b(n) << ' '  << b(n+1) << endl;
  return 0;
}
```

автор: Иван Георгиев

Задача А2 • НАЙ-ДЪЛЪГ ПЪТ

Даден е ориентиран ацикличен граф с положителни целочислени тегла (дължини) по дъгите, т. е. зададена е съвкупност от върхове, като за някои двойки различни върхове е зададена по една насочена дъга от единия към другия връх. Върховете са номерирани от 1 до N. Път от връх 1 до връх j, $1 < j \leq N$, наричаме последователност от дъги, такива че движейки се последователно по тях (спазвайки посоката им), можем да преминем от връх 1 до връх j, като не повтаряме нито дъги, нито върхове. Ацикличността на графа означава, че не е възможно движейки се по дъги, да се върнем в началния връх. Напишете програма `LONG`, която извежда дължината на най-дългия път, започващ от връх 1 и завършващ в някой връх от графа.

Вход

От първия ред на входа програмата прочита броя на върховете N и броя на дъгите M. Следват M реда, всеки съдържащ номерата на началото и края на поредната дъга и дължината ѝ. Числата във всеки от редовете са разделени с по един интервал.

Изход

На един ред на стандартния изход се извежда дължината на намерения път.

Ограничения:
$1 < N < 1000$
$0 < \textit{тегла на дъгите} \leq 999$

Пример

Вход	Изход
5 8 1 2 5 1 4 4 1 5 1 2 3 3 2 5 1 4 3 2 5 3 3 5 4 2	10

РЕШЕНИЕ

Прилагаме топологична сортировка върху графа, с което преномерираме върховете така, че ако съществува дъга от връх i до връх j, да е изпълнено, че $i < j$. При това, върхът с номер 1 приема номер i, като изобщо $i \neq 1$. Но лесно може да съобразим, че тогава върховете с номера, по-малки от i не е нужно да бъдат разглеждани по-нататък. Премахваме тези върхове и означаваме с N броя на останалите. Така по-нататък считаме, че върхът с номер 1 запазва номера си след топологичната сортировка. При дадената в условието на задачата максимална стойност за N, трябва да бъде реализиран алгоритъм за топологична сортировка с времева сложност от порядъка на N^2, за да може програмата да работи в рамките на определеното ѝ време.

Нека с $a[i][j]$ е означена дължината на дъгата, излизаща от връх i и влизаща във връх j. В случаите, когато не съществува дъга от i към j, считаме, че в $a[i][j]$ е записано отрицателно число с голяма абсолютна стойност.

Започваме постъпково да строим дърво, като включваме последователно към дървото върховете на графа според реда, получен от топологичното сортиране: $x_1 = 1, x_2, x_3, \ldots, x_N$. За всеки включен връх x_i записваме стойност $t[x_i]$, равна на дължината на най-дългия път от връх x_1 до връх x_i. В началото полагаме $t[x_1]= 0$. Нека на i-тата стъпка сме построили дърво, съдържащо върховете $x_1, x_2, \ldots, x_i$, и сме пресметнали стойностите $t[x_1]$, $t[x_2]$, $\ldots$, $t[x_i]$. Следващият връх, който трябва да включим в дървото е x_{i+1} като наследник на някой друг връх x_j от дървото ($j = 1, 2, \ldots, i$). Индекса j избираме така, че да съществува дъга от x_j до x_{i+1} в графа и сумата $t[x_j] + a[x_j][x_{i+1}]$ да е максимална. Полагаме $t[x_{i+1}] = t[x_j] + a[x_j][x_{i+1}]$.

Накрая, решението на задачата се получава от най-голямата стойност измежду $t[x_1]$, $t[x_2]$, $\ldots$, $t[x_N]$.

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

const int MIN=-999999;
const int N_max=500;
int M,N;
int a[N_max][N_max];

void input()
```

```
{
  cin >> N >> M;
  for(int i=1;i<=M;i++)
  {
    int b,e,w;
    cin >> b >> e >> w;
    a[b][e]=w;
  }
}

int u[N_max], v[N_max];
void topsort(int b[N_max][N_max])
{
  int a[N_max][N_max];
  for(int i=1;i<=N;i++)
  for(int j=1;j<=N;j++)
   a[i][j]=b[i][j];

  for(int k=1;k<=N;k++)
  {
   int j0;
   for(int j=1;j<=N;j++)
   if(u[j]==0)
   { int f=0;
     for(int i=1;i<=N;i++)
      if(a[i][j]>0)f=1;
     if(f==0) {j0=j;break;}
   }
   u[j0]=k;
   v[k]=j0;
   for(int j=1;j<=N;j++) a[j0][j]=0;
  }
}

int t[N_max];
void tree()
{
  for(int i=1;i<=N;i++)
  for(int j=1;j<=N;j++)
   if(a[i][j]==0) a[i][j]=MIN;
  int j0=1;
  while(v[j0]!=1) j0++;
  t[j0]=0;
  t[j0+1]=a[v[j0]][v[j0+1]];
  for(int j=j0+2;j<=N;j++)
  {
    int m=0;
    for(int k=j0;k<j;k++)
     {
      if(t[k]+a[v[k]][v[j]]>m)m=t[k]+a[v[k]][v[j]];
```

```
      t[j]=m;
     }
  }
  int m=0;
  for(int j=1;j<=N;j++) if(t[j]>m)m=t[j];
  cout << m << endl;
}

int main() {
 input();
 topsort(a);
 tree();
}
```

автор: Емил Келеведжиев

Задача А3 • ШОКОЛАДЕНА ИГРА

Гошо обича да играе игрички – 3D shooter-и, RPG-та, стратегии .... Не става от компютъра, докато не превърти игра. Разви завидни геймърски умения, наду се и заяви, че няма игра, на която да бъде победен. Гошо не е много умен (като всеки средностатистически геймър) и не подозира, че съществуват и други игри. Пешо е състезател по информатика, който отделя много време да усъвършенства уменията си, не му остава време за игрички, заради което е заклеймен от обществото като ламер. Пешо разбира, че неговият шанс да се отърве от славата си на ламер е, да предизвика Гошо с трудна игра. Гошо няма как да откаже, защото рискува името си на прочут геймър, но и моли да му помогнете (бидейки също заклети геймъри, но малко по-умни), като напишете програма `FGAME`, която да печели играта, предложена от Пешо.

Играта се състои в разчупване на парчета шоколад от различни марки на по-малки парчета. Всеки шоколад е съставен от цяло положително число блокчета – блокче е най-малката неделима единица шоколад :=). Шоколадово парче може да се разчупи на няколко по-малки парчета само така, че всяко новополучено парче е съставено от цяло положително число блокчета. Играта започва с C шоколадови парчета, по едно от C различни марки. За шоколадово парче от i-тата марка е позволено да се разчупва на точно S_i, $i = 1, 2, \ldots, C$ по-малки парчета, състоящи се от ненулев брой блокчета. Играчите се редуват да разчупват по едно парче шоколад, като първи е Гошо. Играта губи този, който не може да направи ход.

Вход

От първия ред на стандартния вход се въвежда броят C на марките шоколад. На втория ред, разделени с по един интервал, са зададени числата S_i, $i = 1, 2, \ldots, C$. По същия начин в третия ред са зададени числата N_i, $i = 1, 2, \ldots, C$, където N_i е броят на блокчетата в парчето шоколад от съответната марка).

Изход

Ако началната позицията е губеща (каквото и да играе Гошо, Пешо може да го победи) програмата трябва да изведе на един ред на стандартния изход -1. Ако позицията е печеливша (съществува стратегия на игра, която може да осигури победа на Гошо, каквото и да играе Петър), тогава програмата трябва да изведе един печеливш първи ход на Гошо. Ако съществуват няколко печеливши първи хода, програмата трябва да изведе този, при който се разчупва парче с най-малък номер на

марката (марките са номерирани от 1 до N, в реда по който съответните им S_i и N_i са зададен във входния файл). Ако съществуват няколко печеливши хода за избраното парче, програмата трябва да изведе този, при който редицата от получените след разчупването парчета, сортирана в нарастващ ред, е най-малка лексикографски. Печелившият ход трябва да се изведе на един ред. Редът започва с марката j на парчето, което трябва да се разчупи, следвана от S_j положителни цели числа – големините на новополучените парчета в брой блокчета, подредени в нарастващ ред (сборът на тези S_j числа трябва да е точно N_i.

Ограничения:
$1 \leq C \leq 15\,000$
$2 \leq Si \leq 15$
$1 \leq Ni \leq 63$

Пример 1

Вход	Изход
2 2 2 6 61	1 1 5

Пример 2

Вход	Изход
2 2 2 1 1	-1

РЕШЕНИЕ

Задачата е типичен пример на комбинаторна игра в граф. Решението се основава върху редица важни резултати от теория на игрите, включително теоремата на Sprague-Grundy (SG-теорема). Тук няма да излагаме цялата теория, а само ще споменем някои по-важни елементи от нея. Подробностите може да намерите в книгата `http://iskren.info/reading/info/algo/Game_theory.pdf`.

Всяка позиция на този вид игри може да се разглежда като връх на ориентиран граф, като две позиции са свързани с ориентирано ребро, ако от едната може да се премине в другата с разрешен от правилата на играта ход. За всяка позиция на играта се изчислява стойност, която за краткост ще наричаме *sg-стойност* на тази позиция. Да наречем позициите от които не може да се извърши ход (т.е. от съответния връх на графа няма излизащо ориентирано ребро) *терминални*. Тези позиции очевидно са губещи. На тях присвояваме sg-стойност 0. За всяка от останалите позиции дефинираме sg-стойност като най-малкото неотрицателно число, което НЕ СЕ СРЕЩА сред sg-стойностите на позициите, достижими с един ход от дадената. Очевидно, освен терминалните, в играта ще има и други губещи позиции. В сила е следното:

Твърдение 1. *За всяка губеща позиция sg-стойността е равна на 0, а за всяка печеливща позиция sg-стойността е различна от 0.*

Сума на игри ще наричаме такава игра, която е съставена от няколко по-малки игри, нямащи нищо общо помежду си. В нашия случай разчупването на едно отделно взето парче шоколад е игра, която не зависи от разчупванията на другите парчета шоколад. Затова разглежданата игра е сума на C по-малки игри. Нека разгледаме сума от C игри с sg-стойности на отделните по-малки игри sg_1, sg_2 ... sg_C съответно, а със sg_0 да означим sg-стойностите за играта сума. В сила е следното:

Твърдение 2. Нека текущата позиция P в играта-сума е представена от вектора $(p_1, p_2, \ldots, p_C)$, където p_i е текущата позиция на i-тата съставяща игра. Тогава

$$sg_0(P) = sg_1(p_1) \oplus sg_2(p_2) \oplus \ldots \oplus sg_C(p_C)\,.$$

Тази много важна формула ни позволява да пресмятаме sg-стойността на позиция в игра, съставена от други игри, както е в нашия случай (операцията $\oplus$ е побитово събиране по модул 2 или изключващо или).

Забележете, че ако след разчупване на парче шоколад на няколко по-малки парчета разглеждаме всяко от по-малките парчета като отделна игра, тогава разчупеното

парче е сума на получените игри. Това ни позволява да приложим Твърдение 2 за да намерим sg-стойност за всяко възможно парче шоколад от избрана марка. Парчетата от марката i с по-малко от S_i блока очевидно са терминални позиции и ще получат sg-стойности 0. Да допуснем, че вече сме намерили sg-стойностите на парчета от i-тата марка с големина $1, 2, \ldots, k-1$. Пресмятаме sg-стойността на парче P с големина k от същата марка по следния начин: разбиваме числото k по всички възможни начини на S_i парчета, колкото е позволеният за марката брой. За всяко от тези разбивания R по формулата от Твърдение 2 намираме sg-стойността за R. От получените sg-стойности на всички възможни разбивания определяме най-малкото неотрицателно цяло число, което НЕ СЕ СРЕЩА измежду тях. Това число ще бъде sg-стойността за парче шоколад P с големина k от i-тата марка.

Вярно е, че марките са много, но възможните видове разбивания са 14 (стойностите на S_i са от 2 до 15). Затова в най-лошия случай е достатъчно да пресметнем sg-стойностите на парчета с големини от 2 до 63, и за разбивания от 2 до 15 парчета (т. е. общо $62 \cdot 14$ възможности). Имайки тези стойности можем да определим за всяко парче от входа (всяка марка) колко е неговата sg-стойност (защото всяко едно от тях е отделна игра). След това трябва да ги съберем по модул 2 за да получим sg-стойността на дадената позиция в цялата игра (тази позиция от която ще играе Гошо). Ако тази sg-стойност е 0 можем да изведем -1 и това е решението на задачата.

Нека сега позицията на Гошо е печеливша. От дефиницията за пресмятането на sg-стойностите ще достигнем и до начина за възстановяване на печелившия ход. От всяка позиция с sg-стойност равна на x има ориентирано ребро (ход) до позиции със sg-стойности $0, 1, 2, \ldots, x-1$. Целта на играча, който се намира в печеливша позиция е да „вкара“ противника в губеща позиция, т. е. такава, за която sg-стойността е равна на 0. Ако sg-стойността на текущата позиция е tsg, то търсим парче шоколад с sg-стойност psg и такова, че правейки ход с него до позиция с sg-стойност nsg, psg $\oplus$ nsg = tsg, т.е nsg = tsg $\oplus$ psg. (Лесно се вижда, че sg-стойността на новополучената игра ще бъде 0). Единственият проблем в случая е, че трябва nsg $<$ psg (за да сме сигурни, че има ход от позицията с sg-стойност psg до позиция с sg-стойност nsg). За целта намираме най-старшия бит в двоичното представяне на числото tsg и търсим такова парче, че в psg същия този най-старши бит е със стойност 1. Това ни гарантира, че tsg $\oplus$ psg е число, по малко от psg, понеже по-старшите битове в резултата ще останат такива, каквито са в psg, а стойността на въпросния бит в резултата ще стане 0, защото е 1 и в двете числа – т. е. в двоичното представяне на числата psg и nsg първата им разлика при сравняване отляво надясно е в този бит, като при psg той е 1, а при nsg е 0. Следователно psg $>$ nsg.

автор: Искрен Чернев

ЗИМНИ МАТЕМАТИЧЕСКИ СЪСТЕЗАНИЯ

Русе, 1-3 февруари 2008 г.

Група В, 9-10 клас

Задача В1 • ЧИСЛОВА РЕДИЦА

Всички естествени числа със сума на цифрите S са подредени в растящ ред. Така например при $S = 3$ се получава редицата 3, 12, 21, 30, 102, 111, 120, 201, 210, 300, ... Напишете програма `SEQ`, която по зададени S и N, намира N-я член на тази редица.

Вход
От един ред на стандартния вход се въвеждат целите числа S и N. Входните данни са такива, че N-ят член на редицата се състои най-много от 100 цифри.

Изход
На един ред на стандартния изход програмата трябва да изведе N-я член на редицата.

Ограничения:
$0 < S < 100$
$0 < N < 10^{15}$

Пример

Вход	Изход
`3 5`	`102`

РЕШЕНИЕ

Една идея за решаването на тази задача е да проверяваме естествените числа, започвайки от 1, дали имат сума на цифрите S. Така последователно намираме всички членове на редицата с номера от 1 до N. По този начин обаче може да решим задачата само за малки стойности на N.

Друга идея за решаване на задачата е следната: първо определяме колко цифрено е търсеното число, след което определяме неговите цифри.

Да означим с $a[s][k]$ броя на членовете на редицата със сума на цифрите s, които имат най-много k на брой цифри ($s = 0, 1, 2, \ldots$). Лесно се съобразява, че $a[s][1] = 1$ за $s = 0, 1, 2, \ldots, 9$ и $a[s][1] = 0$ за $s \geq 10$, както и че $a[0][k] = 1$ за всяко $k \geq 1$. Как да пресметнем $a[s][k]$ при $k \geq 2$? Всеки член на редицата, който има най-много k цифри може да разглеждаме като k-цифрено число, което започва с 0, 1, 2, 3, 4, 5, 6, 7, 8 или 9 и сума на останалите цифри съответно s, $s-1$, $s-2$, $s-3$, $s-4$, $s-5$, $s-6$, $s-7$, $s-8$ или $s-9$. Разбира се, първата цифра може да бъде p само когато $s - p \geq 0$. Следователно

$$a[s][k] = a[s][k-1] + a[s-1][k-1] + \ldots + a[s-9][k-1] \text{ при } s \geq 9\,.$$

При $s < 9$ в сумата влизат само тези събираеми, за които $s - p \geq 0$. От намерената рекурентна зависимост следва, че $a[s][k] = a[s-1][k] + a[s][k-1]$ при $s < 10$ и $a[s][k] = a[s-1][k] + a[s][k-1] - a[s-10][k-1]$ при $s \geq 10$.

Ясно е, че търсеният N-ти член на редицата се състои от k цифри точно тогава, когато $a[s][k-1] < N \leq a[s][k]$.

Като използваме направените разсъждения, лесно може да определим цифрите на търсеното число. Ако търсеното число е едноцифрено, то е равно на s. Ако числото

има повече цифри, то първата му цифра е най-малкото неотрицателно цяло число p, за което $a[s][k-1]+\ldots+a[s-p][k-1] \geq N$. Оставащата неизвестна част се състои от $k-1$ цифри (като може да започва и с 0), има сума на цифрите $s-p$ и е $(N-a[s][k-1]-\ldots-a[s-p+1][k-1])$-ти член на редицата със сума на цифрите $s-p$. Това ни позволява да намерим и останалите неизвестни цифри на числото.

```
#include<iostream>

using namespace std;

int main()
{
    int s;
    long long n;
    long long a[128][128];
    cin >> s >> n;
    for(int i=0; i<=s; i++)
      if (i<10) a[i][1] = 1;
      else a[i][1] = 0;
    int k=1;
    while(a[s][k] < n)
      { k++;
        a[0][k] = 1;
        for(int i=1; i<=s; i++)
          if (i<10) a[i][k] =  a[i-1][k] + a[i][k-1];
          else a[i][k] = a[i-1][k] + a[i][k-1] - a[i-10][k-1];
      }
    for(int k1=k; k1>1; k1--)
      { int p = 0;
        long long f = a[s][k1-1];
        while(f<n)
          { p++;
            f = f + a[s-p][k1-1];
          }
        cout << p ;
        n = n - f + a[s-p][k1-1]; s = s - p;
      }
    cout << s << endl;
    return 0;
}
```

автор: Младен Манев

Задача В2 • РЕБРА И ЦИКЛИ

Даден е неориентиран граф с n върха и m ребра. Напишете програма `EDGE`, която намира броя на ребрата, които не участват в цикъл.

Вход
На първия ред на стандартния вход са записани числата n и m. На всеки от следващите m реда са записани по две числа, представляващи краищата на поредното ребро. Върховете са номерирани с числата от 1 до n.

Изход
Резултатът да се изведе на стандартния изход.

Ограничения:
$1 \leq n \leq 10\,000$, $1 \leq m \leq 20\,000$
В 40% от тестовете $n \leq 1\,000$, $m \leq 2\,000$.

Пример

Вход	Изход
7 8	1
5 1	
1 2	
2 6	
3 4	
4 7	
6 7	
2 5	
3 6	

РЕШЕНИЕ

Ребро, което не участва в нито един цикъл, се нарича мост. Отстраняването на такова ребро увеличава броя на свързаните компоненти в графа.

За да определим дали реброто (u, v) е мост, може да действаме по следния начин: отстраняваме реброто от графа и проверяваме дали има път от u до v, например използвайки търсене в дълбочина. Така, ако графът е представен със списъци на съседите ще получим алгоритъм със сложност (m) за определянето дали едно отделно ребро е мост и със сложност (m^2) за решаване на цялата задача. Тази идея е реализирана в програмата `edge-slow.cpp`. Ако графът е представен с матрица на съседство, сложността за всяко търсене в дълбочина ще бъде (n^2) и за целия алгоритъм (mn^2).

Задачата може да бъде решена с използването само на едно търсене в дълбочина.

Да означим с $pr[u]$ непосредствения предшественик на връх u при търсенето в дълбочина. Ако u е първият посетен връх от една свързана компонента, то $pr[u] = 0$.

Да означим с $A(u)$ множеството от върхове v, за които $pr[v] = u$, т. е. върхът u се явява непосредствен предшественик на v при търсенето в дълбочина. Да означим с $B(u)$ множеството от върхове v, които са съседи на u в графа, но $pr[v] \neq u$ и $pr[u] \neq v$.

Номерираме върховете според реда на посещаването им. Нека за връх u съответният номер да е $num[u]$. За всеки връх u пресмятаме $low[u]$, дефинирано по следния начин:

$$low[u] = \min(num[u];\ low[v] \text{ за всяко } v \text{ от } A(u);\ num[v] \text{ за всяко } v \text{ от } B(u)).$$

Ако за някой връх u се окаже, че $low[u] = num[u]$ и $pr[u] \neq 0$, то реброто $(pr[u], u)$ е мост.

Програмата `edge-fast.cpp` реализира разглежданата идея.

За повече подробности виж кой да е текст за двойно свързани компоненти.

Реализация 1

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

const int NMAX = 10000+10;
const int MMAX = 20000+10;

struct edge { int u,v; };

edge e[MMAX];
vector<int> a[NMAX];
int visited[NMAX];
int n, m, cnt;

void dfs(int u)
{  visited[u] = 1;
   for(int i=0; i<a[u].size(); i++)
     if(visited[a[u][i]] == 0)
       dfs(a[u][i]);
}

void remove(int u, int v)
{  int i=0;
   while(a[u][i] != v) i++;
   a[u][i] = a[u][a[u].size()-1];
   a[u].pop_back();
   i=0;
   while(a[v][i] != u) i++;
   a[v][i] = a[v][a[v].size()-1];
   a[v].pop_back();
}

void restore(int u, int v)
{ a[u].push_back(v);
  a[v].push_back(u);
}

int main()
{
   cin >> n >> m;
   for(int i=0; i<m; i++)
   { int u,v;
     cin >> u >> v;
```

```
    e[i].u = u; e[i].v = v;
    a[u].push_back(v);
    a[v].push_back(u);
  }
  cnt = 0;
  for(int i=0; i<m; i++)
  { int u = e[i].u, v = e[i].v;
    remove(u,v);
    for(int i=1; i<=n; i++) visited[i] = 0;
    dfs(u);
    if(visited[v] == 0) cnt++;
    restore(u,v);
  }
  cout << cnt << endl;
  return 0;
}
```

Реализация 2

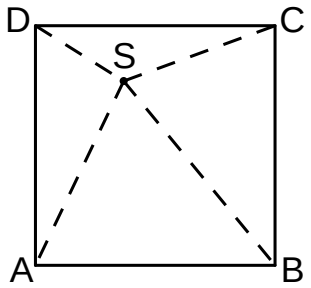
```
#include <iostream>
#include <vector>
using namespace std;

const int NMAX = 10000+10;
vector<int> a[NMAX];
int pr[NMAX], num[NMAX], low[NMAX];
int n,m;
int dfsnum, cnt;

void dfs(int u)
{  dfsnum++;
   low[u] = num[u] = dfsnum;
   for(int i=0; i<a[u].size(); i++)
   { int v = a[u][i];
     if(num[v] == 0)
     { pr[v] = u;
       dfs(v);
       if(low[u]>low[v]) low[u]=low[v];
     }
     else
       if(v!=pr[u] && low[u]>num[v]) low[u]=num[v];
   }
   if(low[u]==num[u] && pr[u]!=0) cnt++;
}

int main()
{  cin >> n >> m;
   for(int i=0; i<m; i++)
   { int u,v;
     cin >> u >> v;
```

```
    a[u].push_back(v);
    a[v].push_back(u);
  }
  dfsnum = 0;
  cnt = 0;
  for(int u=1; u<=n; u++)
    num[u]=0;
  for(int u=1; u<n; u++)
    if(num[u]==0)
    { pr[u]=0; dfs(u); }
  cout << cnt << endl;
  return 0;
}
```

автор: Стоян Капралов

Задача В3 • ПЛАТКА

Завод произвежда специална квадратна платка със страна 100 mm. Ако означим върховете съответно с A, B, C и D (вижте чертежа), то върху платката се нанасят праволинейни пътечки от тях до такава точка S, че сумата от разстоянията $AS + DS$ е зададена константа a, а сумата $AS + BS$ е друга зададена константа b. Проблемът е, че стойностите на a и b се менят често.



Напишете програма `PLATE`, която за въведени a и b определя общата дължина на пътечките върху платката $AS + BS + CS + DS$.

Вход

От стандартния вход се въвежда един ред с двете реални положителни числа a и b, разделени с интервал. Числата са в милиметри, с точност до стотни от милиметъра.

Изход

Изведете на стандартния изход един ред с намерената сума, с точност до стотни от милиметъра.

Ограничения:
$100 < a, b \leq 222$

Пример

Вход	Изход
130 167.08	288.00

РЕШЕНИЕ

В задачата се търси пресечната точка на две елипси S. Директното трасиране с тази точност е с твърде голяма сложност. Предлагаме „клатене“ – отначало „на едро“ (със стъпка 1 е добре), а след това – до желаната точност. За зададените ограничения е добре. При по-големи ограничения това решение е бавно. За любопитните предлагаме и решение с двоично търсене по дъга от елипса.

Реализация 1

```
#include <stdio.h>
#include <math.h>
typedef struct {double x,y;} Point;
double a,b;
Point A={0,0},B={100,0},C={100,100},D={0,100},S;
double dist(Point A,Point B)
{return sqrt((A.x-B.x)*(A.x-B.x)+(A.y-B.y)*(A.y-B.y));
}
void findS(double e)
{double st=1;
 int fl;
 Point P={100,100};
 do
 {fl=0;
  for(S.y=P.y;S.y>0;S.y-=st)
  {for(S.x=P.x;S.x>0;S.x-=st)
   {fl=fabs(dist(D,S)+dist(A,S)-a)<st && fabs(dist(A,S)+dist(B,S)-b)<st;
    if (fl) break;
   }
   if (fl) break;
  }
  P.x=S.x+st;
  P.y=S.y+st;
  st/=10;
 }while(st>e);
}
double findRes(void)
{return a+dist(B,S)+dist(C,S);
}
int main()
{scanf("%lf %lf",&a,&b);
 findS(0.0001);
 printf("%.2lf\n",findRes());
 return 0;
}
```

Реализация 2

```
#include <stdio.h>
#include <math.h>
const double s=100;
typedef struct {double x,y;} Point;
typedef struct {Point O;double R,r;} Ellipse;
double a,b;
Point A={0,0},B={s,0},C={s,s},D={0,s},O1={s/2,0},O2={0,s/2},S;
Ellipse e1,e2;
double dist(Point A,Point B)
{return sqrt((A.x-B.x)*(A.x-B.x)+(A.y-B.y)*(A.y-B.y));}
```

```
double ToEllipse(Point A,Ellipse e)
{return (A.x-e.O.x)*(A.x-e.O.x)/(e.R*e.R)+(A.y-e.O.y)*(A.y-e.O.y)/(e.r*e.r);
}
void findS(double e)
{double t,l,r,m;
 l=acos(-s/b);
 r=0;
 do
 {m=(l+r)*0.5;
  S.x=e1.O.x+e1.R*cos(m);
  S.y=e1.O.y+e1.r*sin(m);
  t=ToEllipse(S,e2);
  if(fabs(t-1)<e) return;
  if (t>1) r=m; else l=m;
 }while(1);
}
double findRes(void)
{return a+dist(B,S)+dist(C,S);
}
int main()
{scanf("%lf %lf",&a,&b);
 e1.O=O1;
 e2.O=O2;
 e1.R=b/2;
 e1.r=0.5*sqrt(b*b-s*s);
 e2.R=0.5*sqrt(a*a-s*s);
 e2.r=a/2;
 findS(0.0001);
 printf("%.2lf\n",findRes());
 return 0;
}
```

автор: Павлин Пеев

ЗИМНИ МАТЕМАТИЧЕСКИ СЪСТЕЗАНИЯ

Русе, 1-3 февруари 2008 г.

Група С, 7-8 клас

Задача С1 • ГАДАТЕЛ

Прочута със своите умения и спечелила доверието на русенци, известната русенска гадателка Руска Русева, след дълги размисли как да привлече вниманието на повече хора, стигнала до извода, че е наложително да използва иновационни методи в работата си. В това динамично време на стремглаво развиващи се информационни технологии, тя не можела да продължава да гадае по традиционните остарели начини. Затова си купила компютър, който бил абсолютно необходим за съвременото модерно гадаене. Базирайки се на многогодишния си опит, след многодневни изчисления, сметки и консултации със звезди, книги и други гадателски пособия, тя най-накрая открила търсената формула, предсказваща бъдещето.

Новата схема за гадене била следната: На всеки човек се предоставя уникалната възможност сам да въведе 2 магически числа с не по-малко от 18 цифри, които ще определят бъдещето му.

Гадателката сама въвежда цяло двуцифрено число, представляващо годините на съответния човек.

Всичко останало е в ръцете на чудната машина наречена компютър, която трябва да изчисли резултатното магическо число и магическата цифра, следвайки следните стъпки: 1) Изчислява се разликата на двете магически числа, като от по-голямото се изважда по-малкото. Получената разлика се умножава с годините. Получава се резултатното магическо число. 2) От полученото резултатно магическо число се определя магическата цифра. Магическата цифра е най-често срещаната цифра от цифрите на резултатното магическо число, която е делител на сумата на цифрите на резултатното магическо число. Ако няколко цифри, делители на сумата на цифрите на резултатното магическо число, се срещат равен брой пъти и това е максималния брой срещания, то магическа е цифрата, срещаща се в по-младшите разряди. Ако няма цифра делител на сумата, то магическо е числото 0.

И ... време е да разберете какво ви очаква:

0 – Веселие; 1 – Щастие; 2 – Неприятности; 3 – Късмет; 4 – Болест; 5 – Успех; 6 – Нещастие; 7 – Любов; 8 – Радост; 9 – Здраве

Но амбициозната гадателка никога досега не е работила с компютър и няма ни най-малка представа от програмиране. Тя ви моли да напишете програма `FUTURE`, която да използва в работата си, като следвате горната схема. По този начин и вие може да видите какво ви очаква!

Вход

На първия ред на стандартния вход се въвежда цяло число N – първото магическо число. На втория ред на стандартния вход се въвежда цяло число M – второто магическо число. На третия ред се въвежда цяло двуцифрено число – годините на човека.

Изход
На първия ред на стандартния изход програмата трябва да изведе цяло число, което е резултатното магическо число. На втория ред на стандартния изход трябва да се изведе цифра, представляваща магическата цифра.

Ограничения:
$10^{17} < N < 10^{200}$
$10^{17} < M < 10^{200}$

Пример

Вход	Изход
111111111111111111 222333444555666777 10	1112223334445556660 3

РЕШЕНИЕ

Програмата реализира метода за преобразуване на дълго число от низ в масив от цели числа. Магическите числа се въвеждат като масиви от символи и се записват в масивите `a` и `b`. Масивите от цели числа са декларирани като глобални променливи и затова се зареждат автоматично с нулеви стойности. Следва запис на отделните цифри в масивите от цели числа `a1` и `b1`. Елементите с нулеви номера от тези масиви съдържат младшите цифри. Номерът на най-старшите цифри на двете дълги числа се пази съответно в променливите `n` и `m` (броят на цифрите е с единица по-голям от стойностите в тези променливи, защото броенето на елементите на масивите започва от нула).

Определяме по-голямото от двете дълги числа, като: 1) сравняваме броя на цифрите – логично е, че числото с повече цифри е по-голямо; 2) ако двете числа имат равен брой на цифрите, то сравняваме цифрите им.

Пресмятаме разликата между по-голямото и по-малкото число, като следваме обичайния „ръчен метод“ за изваждане на две многоцифрени числа. Особено внимание трябва да се обърне на случаите, когато цифрата, от която вадим е по-малка е от тази, която трябва да извадим от нея. В тези случаи е необходимо да се вземе едно от цифрата в по-старшия разряд.

Следва умножение на елементите на получения с разликата масив с с двуцифреното число, като се спази правилото за умножение на многоцифрени числа. Елементите на масива с се умножават с всяка една от двете цифри на двуцифреното число. Получените две произведения, записани в масивите с цели числа `a2` и `b2` се събират.

Изчисляваме сумата на цифрите на полученото произведение и колко пъти се среща всяка цифра, която е делител на сумата. След това определяме коя е най-често срещаната цифра, делител на сумата на цифрите на съответното дълго число.

```
#include<iostream>
using namespace std;
char a[200], b[200];
int a1[200], b1[200],a2[200], b2[200], c1[200], c[200], d[200],a11[200], b11[200];
int sum, k, k1, k2, k3, n, n1, n2, p, m,i,maxi, maxi1, j;
int main() {
  cin>>a>>b>>k;
  n=strlen(a); m=strlen(b);
  if (n>m) n1=n; else n1=m;
  for (i=n-1; i>=0; i--) a1[n-1-i]=a[i]-'0';
  for (i=m-1; i>=0; i--) b1[m-1-i]=b[i]-'0';
```

```
i=0; maxi1=0; maxi=0;
do {
  if ((a1[n1-1-i]>b1[n1-1-i])) maxi=1;
  else if (a1[n1-1-i]!=b1[n1-1-i]) maxi1=1; else i=i+1;
} while (maxi==0 && maxi1==0);
if (maxi1==1) {for (i=0; i<n1; i++) swap(a1[i],b1[i]);}
for (i=0; i<n1; i++) {
  if (b1[i]>a1[i]) {a1[i]=a1[i]+10; a1[i+1]=a1[i+1]-1;}
  c[i]=a1[i]-b1[i];
}
k1=k%10; k2=k/10; n2=n1;
for (p=0,i=0; i<n1; i++) {
  k3=c[i]*k1;
  if (k3>9) {
    a2[i]=k3%10+p;
    if (i==n1-1) {a2[i+1]=k3/10; n2=n1+1;} else p=k3/10;
  } else {a2[i]=k3+p; p=0;}
}
for (p=0,i=0; i<n1; i++) {
  k3=c[i]*k2;
  if (k3>9) {
    b2[i+1]=k3%10+p;
    if (i==n1-1) {b2[i+2]=k3/10; n2=n1+1;} else p=k3/10;
  }
  else {b2[i+1]=k3+p; p=0;}
}
for (p=0,i=0; i<=n2; i++) {
  k3=a2[i]+b2[i];
  if (k3>9) {
    d[i]=k3%10+p;
    if (i==n2) {d[i+1]=k3/10; n2=n2+1;} else p=k3/10;
  } else {d[i]=k3+p; p=0;}
}
j=0; i=n2;
do {if (d[i]==0) j=j+1; i=i-1;} while (d[i]==0);
sum=0;
for (i=n2-j; i>=0; i--) {cout<<d[i]; sum=sum+d[i];}
cout<<endl;
k=0;  k2=0;
for (i=0; i<=n2; i++)
  if (d[i]>0 && sum%d[i]==0) {
    k1=d[i];
    if (i>0)
      for (j=0;j<k;j++)
        if (a11[j]==k1) {b11[j]=b11[j]+1; k2=1;}
    if (k2==0) {a11[k]=k1; b11[k]=1; if (i<n2) {k=k+1;}}
    k2=0;
  }
for (i=0; i<k-2; i++)
  if (b11[i]<b11[i+1]) {swap(b11[i],b11[i+1]); swap(a11[i],a11[i+1]);}
```

```
  cout<<a11[0]<<endl;
  return 0;
}
```

автор: Сюзан Феимова

Задача С2 • ТРИЪГЪЛНИЦИ

Някъде в близкото бъдеще всички олимпиади по природо-математическите дисциплини ще се провеждат по едно и също време и на едно и също място, както олимпийските игри по спортните дисциплини. България, като пръв организатор на Олимпиадите по информатика, може да претендира за домакин на такава Суперолимпиада. Затова привърженици на идеята у нас започнали да правят проект на бъдещото Олимпийско градче. В плана на градчето има N площада, номерирани с числата от 1 до N, като някои двойки площади са свързани с алеи – броят на алеите е M. Когато планът бил готов, започнали да се измислят имена на площадите и алеите. В този момент на някои млади информатици, които се готвят за Балканиадата, им хрумнало, че няма да е зле една част на градчето да бъде посветена на олимпиадите по математическите дисциплини. Затова си задали въпроса, дали в плана няма да се намери триъгълник – три площада, всеки два от които са свързани с алея – за да може тези три площада да се кръстят Площад на математиката, Площад на информатиката и Площад на математическата лингвистика. Напишете програма `TRI`, която да проверява дали има такива три площада.

Вход
На първия ред на стандартния вход ще бъдат зададени целите положителни числа N и M. Всеки от следващите M реда ще съдържа номерата на два площада, свързани с алея.

Изход
На единствения ред на стандартния изход програмата трябва да изведе сумата от трите номера на площади, които образуват триъгълник. Ако има повече от един триъгълник в плана, програмата трябва да изведе най-малката сума от номера на площади, които образуват триъгълник. Ако в плана няма триъгълници, програмата трябва да изведе на единствения ред на стандартния изход числото 0.

Ограничения:
$3 < N < 2000$
$3 < M < 20000$

Пример 1

Вход	Изход
5 7	10
1 5	
1 4	
4 5	
2 3	
2 5	
3 4	
3 5	

Пример 2

Вход	Изход
6 7	0
1 2	
2 3	
3 4	
4 5	
5 6	
6 1	
5 2	

РЕШЕНИЕ

Разглеждаме плана на града като краен неориентиран граф, върховете на който са площадите, а ребрата – алеите на градчето. Възможни са различни решения, които разбира се имат и различна алгоритмична сложност.

Най-бавно е следното решение. Поставяме достатъчно голямо число (при номера от 1 до 1999 едно възможно число е 6000, например) в променливата `S` за начална стойност на минимума на сумата. След като се въведат ребрата в масив `G` с `M` реда и 2 стълба се генерират всички тройки от ребра (`i`,`j`,`k`) с тройния цикъл

```
for (i=1;i<=M-2;i++)
  for (j=i+1;i<=M-1;j++)
    for (k=j+1;k<=M;k++)
```

След това се образува множеството от краищата `A` = {`ui`,`vi`,`uj`,`vj`,`uk`,`vk`} на ребрата от (`i`,`j`,`k`) и се проверява колко са елементите на това множество. Ако са точно 3, т.е. `A` = `u`,`v`,`w`, тогава и само тогава трите ребра образуват триъгълник. Сравняваме сумата `u+v+w` с текущата стойност на `S` и ако е по-малка, запомняме в `S` сумата. В края на алгоритъма, ако стойността на `S` е началната, значи не е намерен триъгълник и извеждаме 0. В противен случай извеждаме стойността на `S`. Сложността на такова решение очевидно е $O(M^3)$ и такъв алгоритъм е неприемлив за големи стойности на M.

Задачата можем да решим и по друг начин. Да въведем графа в матрица на съседства, за да можем лесно да проверяваме дали два върха са свързани с ребро. По подобен на първия алгоритъм начин, да генерираме всички тройки от върхове `B` = `u`,`v`,`w`. Проверката дали върховете от `B` образуват триъгълник, когато графът е представен с матрица на съседства, е елементарна. Намирането на триъгълника с минимална сума на номерата на трите върха можем да направим по същия начин, както в предния алгоритъм. Сложността на такова решение очевидно е $O(N^3)$ и такъв алгоритъм е доста по-добър от първия и все пак ще бъде бавен при големи стойности на N и M.

Да се опитаме да намерим по-добър алгоритъм. За да образува реброто (u, v) на графа триъгълник, трябва да намерим връх w такъв, че (u, w) и (v, w) също са ребра на графа. Нека с G_u да означим множеството от върхове на графа, които са съседни на u, а с G_v – множеството от върхове на графа, които са съседни на v. Ако сечението $C = G_u \cap Gv$ е непразно, тогава всеки връх w от C ще образува триъгълник с u и v. При това не е нужно да се проверяват всички върхове от C, а само този с най-малък номер, тъй като търсим триъгълника с минимална сума от номера на върхове.

И така, да представим графа по два начина, със списъка на ребрата, който въвеждаме директно от входа и с матрица на съседства. За всяко (u, v) от списъка на ребрата да намерим върха с най-малък номер w от C, като последователно сравняваме стойностите от реда на u и реда на v в матрицата на съседства, докато намерим първия връх, за който и в двата реда има стойност 1, след което, ако се налага, да обновим стойността на S до $u + v + w$. Ако не намерим връх, за който и в двата реда има стойност 1, значи сечението C е празно и реброто (u, v) не участва в триъгълник. Сложността на този алгоритъм е $O(MN)$, защото за всяко ребро намирането на сечението C става със сложност $O(N)$. За графи с брой ребра много по-малък от $N(N-1)/2$, какъвто е графът в задачата, той е много по-добър от алгоритъма със сложност $O(N^3)$.

Но това не е всичко, което може да се направи в тази задача по отношение подобряване на сложността на алгоритъма. При определени условия намирането на сечението C може да се направи със сложност много по-малка от $O(N)$. За целта трябва вместо в матрица на съседства второто представяне на графа да стане със списъци на съседите за всеки връх. При това всички списъци на съседите трябва да са сортирани в

нарастващ ред. В този случай намирането на сечението на два сортирани списъка с P и Q елемента, $P \leq Q$, може да се направи със сложност $O(P \log_2 Q)$ с алгоритъм, при който всеки от върховете в по-късия списък се търси двоично в по дългия. Да означим средния брой съседи на връх в графа с D. Очакваната сложност на последния алгоритъм ще бъде $O(N\,D^2 + M\,D\,\log_2 D)$, ако сортираме по елементарен начин и $O(N\,D\,\log_2 D + M\,D\,\log_2 D) = O(M\,D\,\log_2 D)$, ако сортираме с някой бърз алгоритъм. Значи той ще бъде по-добър от третия, ако $D\,\log_2 D < N$, което може да се разбере още с прочитането на N и M, защото $D = 2M/N$, и в зависимост от това да се използва третия или четвъртия алгоритъм. Оставяме на читателя да обмисли подробностите при реализацията на такъв алгоритъм.

За получаване на пълния брой точки в състезанието е достатъчно да бъде реализиран третият алгоритъм. По-долу даваме програма за него.

```
#include <stdio.h>
#define MAXN 2000
#define MAXM 20000
#define MAXSUM 6000
int N,M,H[MAXM][2],G[MAXN][MAXN]={0};

int main()
{
   int i,j,u,v,S;
   scanf("%d %d",&N,&M);
   for(i=1;i<=M;i++)
   {  scanf("%d %d",&H[i][0],&H[i][1]);
      G[H[i][0]][H[i][1]]=1;
      G[H[i][1]][H[i][0]]=1;
   }
   S= MAXSUM;
   for(i=1;i<=M;i++)
   {   j=1;u=H[i][0];v= H[i][1];
       while(j<=N&&G[u][j]*G[v][j]==0)j++;
       if(j<=N&&u+v+j<S)
          S=u+v+j;
   }
   if(S== MAXSUM) printf("0\n");
   else printf("%d\n",S);
}
```

автор: Красимир Манев

Задача С3 • КОНТЕКСТНО ТЪРСЕНЕ

Всички съвременни интернет-търсачки показват резултатите от търсенето в контекст. Например, ако търсим с думата „състезания“, търсачката извежда реда

`Зимните математически състезания ще се проведат в`

откъдето разбираме, че става въпрос именно за зимните математически състезания. Напишете програма `CONTEXT`, която по зададен текст и ключова дума извежда всички нейни срещания, заедно с контекста.

Вход
От първия ред на стандартния вход се въвежда едно число N.

От втория ред се въвежда ключова дума или фраза, която може да съдържа интервал, латински букви, цифри, препинателни и математически знаци.

От третия ред се въвежда текст, който може да съдържа интервали, латински букви, цифри, препинателни и математически знаци.

Ако в началото и края на фразата и текста има интервали, те се игнорират.

Изход
Всяко срещане на ключовата дума или фраза се извежда заедно с контекста си на отделен ред, като сравнението в текста пренебрегва разликата между малки и главни букви. Дължината на всеки ред е равна на $2N +$ дължината на ключовата дума или фраза. За целта от двете страни на ключовата дума или фраза се добавят по N символа. Когато това е невъзможно (думата или фразата е близо до началото или края на текста) за някоя от страните, символите се допълват от другата страна на ключовата фраза до нужния размер. В случай, че текстът е по-кратък от $2N +$ дължината на ключовата дума или фраза, да се изведе целият текст.

Ограничения:
$0 \leq N \leq 100$
Ключова дума – не повече от 50 символа
Текст – не по-дълъг от 65 000 символа

Пример

Вход
```
8
␣sNoW
␣␣␣Snow is a type of precipitation in the
form of crystalline water ice, consisting
of a multitude of snowflakes that fall from
clouds.  Since snow is composed of small
ice particles, it is a granular material.
Many winter sports, such as skiing and
snowboarding depend on snow...␣␣␣
```

Изход
```
Snow is a type of pr
tude of snowflakes t
. Since snow is comp
ing and snowboarding
ng depend on snow...
```

РЕШЕНИЕ

Задачата е чисто практическа и се състои в съобразяването на няколко ключови момента:

1. Динамични масиви

В `string` могат да се запазят едва до 255 символа. При повечето компилатори в статичен масив също не може да бъде побран голям текст. Ето защо масивът трябва да бъде създаден динамично по следния начин:

```
char *InputText = new char [MaxSize_InputText];
```

2. Бързо изчистване на празните интервали.

Изключително бавно е, ако трябва да се прехвърлят елементите на целия масив за всеки празен интервал в началото. Единият вариант е да се игнорират всички символи в началото, докато се стигне до такъв, различен от интервал.

Вторият вариант е да се използва ново адресиране на масива в паметта (в C++ това

става чрез `memmove` и `memset`).

3. Търсене на подниз, без значение на малки и главни букви.

При правилен избор на структура, операторът `strnicmp` е бърз и лесен начин това да се постигне.

4. Избягване на преизчисляването на едни и същи стойности.

Типичен пример за допълнително забавяне е, ако програмата всеки път преизчислява дължината на голям низ. Ако дължините на примерния текст и ключовата дума (фраза) се пазят в променлива, това също ускорява работата на програмата.

5. Точно изчисляване на мястото, откъдето да се изведат $2N +$ *дължина на ключовата дума или фраза* символа.

Нека за краткост наричаме дължината, изчислена на горния ред, „обща дължина".

Ако общата дължина надвишава тази на текста, се извежда целия текст. В противен случай, можем да изчислим изместване $Delta$, което в общия случай е 0.

Нека i е позицията, на която е открита ключовата дума. В такъв случай:

- Ако $N - I > 0$, то $Delta = N - i$;
- Ако $I + N +$ *дължината на ключовата дума* $>$ *дължината на текста*, то $Delta =$ *дължината на текста* $-$ *дължината на ключовата дума* $- i - N$.

След това остава само да изведем низ с дължина общата дължина, започвайки от $i - N + Delta$ позиция в текста.

```
#pragma hdrstop
#pragma argsused
#include <stdio.h>
#include <memory.h>
#include <string.h>
using namespace std;
#define MaxSize_Keyword 50
#define MaxSize_InputText 65000
void RemoveSpaces (char *X)
// Removes blank spaces by just moving memory borders
{
  int firstChar = 0;
  int length = strlen(X);
  while (X[firstChar]==' ') firstChar++;
  if (firstChar>0) memmove(X,X+firstChar,length-firstChar + 1);
  length -= firstChar;
  if (length == 0) return;
  int lastChar = length-1;
  while (X[lastChar]==' ' && lastChar>0) lastChar--;
  if (length-lastChar-1 > 0 ) memset(X+lastChar+1,0,length-lastChar-1);
}
int main()
{
  // Variables Used
  int N;
  char *Keyword = new char [MaxSize_Keyword];
```

```
  char *InputText = new char [MaxSize_InputText];
  // Read Data
  scanf("%d\n",&N);
  gets(Keyword);
  gets(InputText);
  // Delete Empty Characters
  RemoveSpaces(Keyword);
  RemoveSpaces(InputText);
  // Calculation
  int KeywordLength=strlen(Keyword);
  int ITLength=strlen(InputText);
  for (int i=0;i<ITLength-KeywordLength+1;i++)
    if (strnicmp(InputText + i, Keyword, KeywordLength)==0)
      {
        if ((KeywordLength+N*2)>=ITLength)
          printf(InputText);
        else
          {
            // Calculate offset
            int Delta=0;
            if ((N-i)>0)
              Delta=N-i;
            else
              if ((i+N+KeywordLength)>ITLength)
                Delta=-i-N-KeywordLength+ITLength;
            fwrite(InputText+i-N+Delta,KeywordLength+N*2,1,stdout);
          }
        printf("\n");
      }
  // Finalization
  delete[] InputText;
  delete[] Keyword;
  return 0;
}
```

автор: Петър Събев

ЗИМНИ МАТЕМАТИЧЕСКИ СЪСТЕЗАНИЯ

Русе, 1-3 февруари 2008 г.

Група D, 6 клас

Задача D1 • КНИГИ

В книжарница са доставени N на брой книги. Всяка книга се кодира с уникален код K – цяло положително число. Собственикът на книжарницата е суеверен човек и смята, че продажбите ще бъдат по-големи, ако изпълни следното правило при подредбата на книгите: първо се подреждат книгите, чиято сума от цифрите на кода е четно число, в нарастващ ред според произведението на цифрите в кода; след това се нареждат книгите, чиято сума на цифрите на кода е нечетно число, в намаляващ ред според произведението на цифрите в кода. Ако два кода имат равно произведение на цифрите си, по-напред ще бъде числото, което е по-малко.

Напишете програма `BOOKS`, която подрежда кодовете на доставените книги, като се спази правилото на собственика.

Вход
На първия ред на стандартния вход се въвежда числото N – брой на доставените книги. На следващите N реда на стандартния вход се въвеждат по едно цяло положително число K, представляващо кода на поредната книга.

Изход
На един ред на стандартния изход програмата трябва да изведе кодовете на книгите, разделени с интервал и подредени според изискванията.

Ограничения:
$1 < N < 30$
$0 < K \leq 2000$

Пример

Вход	Изход
4 22 23 45 33	22 33 45 23

РЕШЕНИЕ

Използвани променливи:

`a` – масив от цели числа, в който записваме въведените кодове на книгите;

`c` – масив от цели числа, в който записваме сумите от цифрите на кодовете със съответни индекис в масива `a`;

`c1` – масив от цели числа, в който записваме произведенията от цифрите на кодовете;

`d` – масив от цели числа, в който записваме кодовете с четна сума на цифрите;

`d1` – масив от цели числа, в който записваме произведенията на цифрите на кодовете с четна сума на цифрите;

`f` – масив от цели числа, в който записваме кодовете с нечетна сума на цифрите;

`f1` – масив от цели числа, в който записваме произведенията на цифрите на кодовете с нечетна сума на цифрите;

При въвеждане кодовете на книгите като поредни елементи на масива `a` изчисляваме сумата и произведението от цифрите на всеки код и ги запазваме съответно в масивите `c` и `c1`, като кодът в масива `a` има съответно сума и произведение в масивите `c` и `c1`, с еднакви индекси.

В отделни масиви `d`, `f`, `d1`, `f1` запазваме съответно кодовете с четни и нечетни суми и съответно произведенията на кодовете с четна сума на цифрите и произведенията на кодовете с нечетна сума на цифрите. Следва сортиране в нарастващ ред на масива `d1`, съдържащ произведенията на цифрите на кодовете с четни суми на цифрите, като едновременно с това разместваме и елемнтите в масива `d`, съдържащ кодовете с четна сума на цифрите. Аналогично постъпваме и с произведенията на цифрите на кодовете с нечетни суми на цифрите, но този път сортираме в намаляващ ред. Остава само да изведем подредените според изискванията кодове.

```
#include<iostream>
using namespace std;
int a[31],c[31],d[31],f[31],c1[31],d1[31],f1[31];
int n, w1, i, j, k, w;
int main()
{
 cin>>n;
 for(i=0; i<n; i++)
    {cin>>a[i];
     c[i]=0; c1[i]=1;
     int w=a[i];
     while(w>0)
        {c[i]=c[i]+(w%10); c1[i]=c1[i]*(w%10);
         w=w/10;
        }
     }
 j=0; k=0;
 for(i=0; i<n; i++)
    {if(c[i]%2==0)
       {d[j]=a[i]; d1[j]=c1[i];j=j+1;}
       else {f[k]=a[i];f1[k]=c1[i];k=k+1;}
    }
 for(i=0;i<j-1;i++)
    for(n=i+1; n<j; n++)
       {if((d1[i]>d1[n])||((d1[i]==d1[n]) && (d[i]>d[n])))
          {
           w=d[i];d[i]=d[n];d[n]=w;
           w=d1[i];d1[i]=d1[n];d1[n]=w;
          }
       }
 for(i=0;i<j;i++) cout<<d[i]<<" ";
 for(i=0;i<k-1;i++)
    for(w1=i+1; w1<k; w1++)
       {if((f1[i]<f1[w1])||((f1[i]==f1[w1]) && (f[i]>f[w1])))
```

```
          {
           w=f[i];f[i]=f[w1];f[w1]=w;
           w=f1[i];f1[i]=f1[w1];f1[w1]=w;
          }
       }
 for(i=0;i<k;i++) cout<<f[i]<<" ";
 cout<<endl;
 return 0;
}
```

автор: Сюзан Феимова

Задача D2 • МЬОБИУС

Знаем, че всяко цяло положително число, което е по-голямо от единица, може да се представи като произведение на прости множители, например $10 = 2 \cdot 5$ и $63 = 3 \cdot 3 \cdot 7$. Виждаме, че при някои представяния простите множители са различни, а при други един и същ прост множител се среща повече от веднъж. Приемаме, че самите прости числа се представят като „произведение“ само от едно число – себе си.

Преди повече от един век големият немски математик Мьобиус изследвал тези представяния и въвел функцията $\mu(n)$, която за всяко цяло положително число n приема една от трите стойности: 0, 1 и -1, съгласно следните правила. Ако $n = 1$, полагаме $\mu(n) = 1$. Ако n има в представянето си поне един повтарящ се прост множител, полагаме $\mu(n) = 0$; в останалите случаи, ако n се представя като произведение от различни прости множители, които са четен брой, полагаме $\mu(n) = 1$, и ако n се представя като произведение от различни прости множители, които са нечетен брой, полагаме $\mu(n) = -1$.

Например $\mu(4) = 0$, $\mu(5) = -1$, $\mu(6) = 1$.

Напишете програма `MOB`, която пресмята стойностите на функцията на Мьобиус $\mu(n)$, когато n пробягва последователно всички цели числа в даден интервал от a до b (включително и краищата a и b).

Вход

От стандартния вход се въвеждат целите числа a и b, разделени с една празна позиция.

Изход

На стандартния изход програмата трябва да изведе стойностите на функцията $\mu(n)$, които съответстват на последователните стойности на n от a до b. Всяка стойност на функцията трябва да бъде изведена на нов ред.

Ограничения:

$0 < a \leq 10\,000$
$0 < b \leq 10\,000$
$a \leq b$

Пример

Вход	Изход
4 7	0 -1 1 -1

РЕШЕНИЕ

Функцията на Мьобиус се пресмята в програмата от функцията `mob`, която връща една от трите възможни стойности. Докато разлагаме числото `n` на прости множители, в променливата `b` получаваме кратността на всеки от множителите, а с променливата `k` броим множителите с кратност 1. Булевата променлива `ok` ни показва дали всички множители имат кратност 1. Ако е така, проверяваме четността на `k`, за да определим стойността, която функцията връща.

```
#include <iostream>
using namespace std;
int mob(int N)
 {
  if(N==1) return 1;
  int k=0;
  bool ok=0;
  int p=2;
  while(p<=N)
   {
    if(N%p==0)
     {
      N /=p;
      int b=1;
      while(N%p==0) {b++; N /=p;}
      if(b==1) k++; else ok=1;
     }
    p++;
   }
  if(!ok)
   {
    if(k%2==0) return 1; else  return -1;
   }
  return 0;
 }
int main()
{
  int a,b; cin>>a>>b;
  for(int i=a;i<=b;i++) cout<<mob(i)<<endl;
}
```

автор: Зорница Дженкова

Задача D3 • ИЗЛИШНИ ЧИСЛА

Кодирано съобщение се състои от числа, всяко от които се среща точно K пъти. Но при предаване на съобщението често в него попадат и други числа, които не са част от оригиналното съобщение и са излишни. Те се срещат по-малко от K пъти. Напишете програма `EXCESS`, която намира броя на излишните числа в дадено съобщение.

Вход

От първия ред на стандартния вход се въвеждат две цели числа: N – брой на

числата в получено съобщение и K – брой на повторенията на числата от коректното съобщение. От следващите N реда се въвеждат числата от съобщението, които са цели положителни и не по-големи от 100 000.

Изход
На стандартния изход се извежда едно цяло число – брой на излишните числа в съобщението.

Ограничения:
$1 \leq N \leq 100\,000$
$2 \leq K \leq 1000$

Пример

Вход	Изход
10 3 7 10 7 4 2 7 4 7 10 4	2

РЕШЕНИЕ

За да определим числата, които се срещат по-малко от K пъти, дефинираме масив `A` със 100 000 елемента, съответстващи на всяко от числата, които могат да участват в съобщението. Зануляваме елементите на този масив. При въвеждане на числата от съобщението пресмятаме броя на срещанията на всяко от тях като увеличаваме с 1 `A[i]`, когато въведем `i`. След това преброяваме елементите на масива `A`, които са по-големи от 0 и по-малки от K.

```
#include<iostream>
using namespace std;
int main () {
    int x, b=0, n, k, *A;
    cin>>n>>k;
    A=new int[100000];
    memset(A, 0, 100000*sizeof(int));
    for(int i=0;i<n;i++) {
        cin>>x;
        A[x-1]++;
    }
    for (int i=0;i<100000;i++)
        if (A[i] && A[i]<k) b++;
    cout<<b<<endl;
    return 0;
}
```

автор: Каталина Григорова

ЗИМНИ МАТЕМАТИЧЕСКИ СЪСТЕЗАНИЯ

Русе, 1-3 февруари 2008 г.

Група Е, 4-5 клас

Задача Е1 • ФИБОНАЧИ

Нашият приятел Умко обичал да решава задачи, в които да открива закономерности за намиране на числа от дадена числова редица. Един ден, решавйки такива задачи, научил, че редицата: $1, 1, 2, 3, 5, 8, 13, 21, \ldots$, където всяко следващо число е сбор от предходните две, се нарича редица на Фибоначи. Да се напише програма `FIB`, която въвежда от клавиатурата число n и намира n-тото число на Фибоначи.

Вход
На единствения ред на стандартния вход се въвежда цяло число n.

Изход
На стандартния изход програмата трябва да извежда едно цяло число, което е n-тото число от редицата на Фибоначи.

Ограничения:
$1 \leq n \leq 1000$

Пример

Вход	Изход
7	13

РЕШЕНИЕ

Първите две числа в редицата на Фибоначи се задават като константи, равни на единица. Организира се цикъл, в който се изчислява сборът на предходнитре две числа. Подготвя се следващият сбор (число), като: (а) второто число от предходния се явява първо число за следващото събиране и (б) полученият сбор е второ събираемо на следвашия сбор.

```
#include<iostream>
using namespace std;
int main() {
  long long int a1=1,a2=1,an,n,i;
  cin>>n;
  if(n==1||n==2) an=1;
  for (i=3;i<=n;i++) {
    an=a1+a2;
    a1=a2;
    a2=an;
  }
  cout<<an<<endl;
  return 0;
}
```

автор: Бистра Танева

Задача Е2 • ВАЛИДНА ДАТА

Един от общоприетите формати за изписване на дата има вида `dd.mm.gggg`, където `dd` е номера на деня, `mm` е номера на месеца, а `gggg` е годината.

За невалидна дата приемаме датата, в която е зададен само грешен номер на ден.

Напишете програма `VALID`, която по зададена дата от 2008 година извежда `Yes`, ако тя е валидна дата и следващата я дата. Ако зададената дата не е валидна, програмата да извежда `No` и броя дни в зададения месец.

Вход
Програмата чете от стандартния вход три цели числа, съответно за ден, месец и година, разделени с един интервал, във вида `dd mm gggg` (годината винаги е 2008).

Изход
Програмата трябва да изведе на стандартния изход два реда. На първия ред се извежда `Yes` или `No`. На втория ред се извежда следващата дата във формат `dd.mm.gggg`, ако въведената дата е валидна и броя на дните във въведения месец в противен случай.

Пример 1

Вход	Изход
`1 2 2008`	`Yes` `2.2.2008`

Пример 2

Вход	Изход
`30 2 2008`	`No` `29`

РЕШЕНИЕ

Трите стойности `d`, `m` и `g` се въвеждат от клавиатурата като цели числа.

За решението на задачата се използва съответствието между номера и броя на дните на месеца, т. е. броя на дните в месеца с номер 1, 3, 5, 7, 8, 10, 12 е 31, като проверяваме условието за въведен грешен номер на деня.

Ако е въведен номер по-голям от 31 се смята, че датата е невалидна. В противен случай датата е валидна и трябва да намерим следващата дата, като, проверяваме дали денят е последен в месеца и месеца не е 12 – тогава търсеният ден е първи и номера на месеца се увеличава с 1, а ако денят е последен и месецът в 2008 г. е 12, тогава годината на следващата дата е 2009. Ако номерът на деня не е последен в месеца, извеждаме стойността му, увеличена с 1.

Аналогично и за месеците съответно с номера 4, 6, 9 и 11.

Тъй като става дума за определена година – 2008, която е високосна, то въведен грешен номер на деня, когато месецът е 2, е този, който е по-голям от 29.

```
#include<iostream>
using namespace std;
int main()
{
    int d,m,g;
    cin>>d>>m>>g;
    if(m==1 || m==3 || m==5 || m==7 || m==8 || m==10 || m==12){
         if(d>31) { cout<<"No"<<endl;
                    cout<<"31"<<endl;
                  }
```

```
        else {
              cout<<"Yes"<<endl;
              if(d==31 && m!=12)cout<<"1."<<m+1<<".2008"<<endl;
              if(d==31 && m==12)cout<<"1.1.2009"<<endl;
              if(d<31) cout<<d+1<<'.'<<m<<".2008"<<endl;
             }
    }
    if(m==4 || m==6 || m==9 || m==11){
          if(d>30) { cout<<"No"<<endl;
                     cout<<"30"<<endl;
                   }
          else {
                cout<<"Yes"<<endl;
                if(d==30) cout<<"1."<<m+1<<".2008"<<endl;
                if(d<31) cout<<d+1<<'.'<<m<<".2008"<<endl;
               }
    }
    if(m==2){
             if(d>29) { cout<<"No"<<endl;
                        cout<<"29"<<endl;
                      }
             else {
                    cout<<"Yes"<<endl;
                    if(d==29) cout<<"1."<<m+1<<".2008"<<endl;
                    if(d<29) cout<<d+1<<'.'<<m<<".2008"<<endl;
                   }
          }
  return 0;
}
```

автор: Мария Енева

Задача Е3 • СТЪПАЛА

Нашият приятел Знайко всеки ден изкачвал няколко стъпала, за да се прибере от училище. Един ден той си задал въпроса: „Как ще изглеждат тези стъпала, ако ги нарисувам на компютъра си?“.

Знайко не бил много добър програмист, затова очаква да напишете програма `STEPS`, която по зададен брой на стъпала N, височина H и широчина L на едно стъпало и символ S начертава стъпалата, както е посочено на примера. Обърнете внимание, че всяко следващо стъпало започва от последния ред на предното.

Вход

От първия ред на стандартния вход се въвеждат N, H и L, разделени с по един интервал. От втория ред се въвежда S.

Изход

На стандартния изход се извеждат начертаните стъпала.

Ограничения:
$0 < N \leq 10$
$1 < H \leq 7$
$0 < L \leq 7$

Пример

Вход
```
4 3 5
+
```

Изход
```
++++++
+    +
+    ++++++
+         +
+         ++++++
+              +
+              ++++++
+                   +
++++++++++++++++++++
```

РЕШЕНИЕ

Всяко стъпало съдържа три различни вида редове:

— *първи ред*, състоящ се от зададения символ, определен брой празни позиции и $L+1$ пъти зададения символ. Само първото стъпало прави изключение от това правило – състои се само от $L+1$ пъти зададения символ;

— *междинен ред*, състоящ се от зададения символ, определен брой празни позиции и зададения символ;

— *последен ред*, състоящ се от $N \cdot L + 1$ пъти зададения символ.

```
#include <iostream>
using namespace std;
int main()
{
  int n,h,l,k,brcol,i,j,sp=0;
  char s;
  cin >>n>>h>>l;
  cin >>s;
  for (i=1;i<=n;i++){
   /* първи ред на стъпало */
   if (i==1) {
       /* първо стъпало */
    for(k=0; k<=l;k++)cout<<s;
   cout<<endl;
   }
   else /* поредно стъпало */
    {
      sp=l*(i-1)-1;
      cout<<s;
      for(k=1; k<=sp;k++) cout<<' ';
      for(k=0; k<=l;k++)cout<<s;
      cout<<endl;}
   /* междинен ред на стъпало */
   if (i==1) sp=l-1;
       else sp+=l;
   for (j=2; j<=h-1;j++) {
```

```
       cout<<s;
       for(k=1; k<=sp;k++) cout<<' ';
       cout<<s;
       cout<<endl;
   }
  }
  /* последен ред */
   brcol=n*l+1;
   for (k=1; k<=brcol; k++)  cout <<s;
   cout <<endl;
}
```

автор: Пламенка Христова