ОЛІМПІАДА 2002 / 2003

8—9 клас

Задача 1. **Порахуй**

Потрібно написати програму, що знаходить всі різноманітні упорядковані пари **A** і **B** за вхідними даними **C** і **D**, такі що **A + B = C**, **A * B = D**, де **A**, **B**, **C**, **D** — цілі числа, що не перевищують за модулем 30000.

Вхідний файл: **RAHUNOK.DAT** у єдиному рядку задано через пропуск два числа — **C** і **D**.

Вихідний файл: **RAHUNOK.ETA** кожний рядок являє собою пари чисел **A** і **B**, розділених одним пропуском. Рядки упорядковані в порядку зростання першого числа пари. У останньому рядку виводиться одне число — кількість знайдених рішень.

Приклад	*Вхідні дані*	*Вихідні дані*
	RAHUNOK.DAT	**RAHUNOK.ETA**
Тест	-3 2	-2 -1 -1 -2 2

Пояснення до розв'язань

Задача є тривіальною повноперебірною задачею, а тому легко розв'язується.

```
Program O_02_03_1;
 var f:text; s,c,d,a,b:integer;
Begin
  {Читання даних з файла}
  assign(f,'rahunok.dat');reset(f);
  readln(f,c,d);close(f);
  s:=0;
  assign(f,'rahunok.eta');rewrite(f);
  For a:=-30000 to 30000 do
   For b:=-30000 to 30000 do
   Begin
    If (a+b=c)and(a*b=d)
    Then Begin
          WriteLN(f,a,' ',b);
          inc(s);
         End;
```

```
    End;
  WriteLN(f,s);close(f);
end.
```

Задача 2. Цифровий корінь

Розглянемо довільне натуральне число і знайдемо суму його цифр, потім суму цифр отриманого числа і так далі, доки не одержимо одноцифрове число. Назвемо це число цифровим коренем.

Потрібно написати програму, що для заданого N ($N < 10^{100}$) знаходить його цифровий корінь.

Вхідний файл: **KORIN.DAT** один рядок, що містить натуральне число **N**.

Вихідний файл: **KORIN.ETA** один рядок, що містить цифровий корінь числа **N**.

Приклад	*Вхідні дані*	*Вихідні дані*
	KORIN.DAT	**KORIN.ETA**
Тест	247	4

Пояснення до розв'язань

Оскільки число може бути дуже велике, організовуємо посимвольне зчитування з файла. Отриманий символ перетворюємо на цифру та відразу накопичуємо суму у змінній S. Далі задача тривіальна: доки чергова сума не стане одноцифровою (< 10), знаходимо суму цифр чергового отриманого числа за допомогою функції **Sum_cifr**.

```
Program O_02_03_2;
Function Sum_cifr(x:word):word;
var S:word;
begin
  S:=0;
  while x<>0 do
  begin
    S:=S+x mod 10;
    x:=x div 10;
  end;
  Sum_cifr:=S;
end;
var f:text; S:word; ch:char;
```

```
Begin
  {Зчитування даних}
  assign(f,'korin.dat'); reset(f);
  S:=0;
  While not eof(f) do
  Begin
   read(f,ch);
   S:=S+(ord(ch)-ord('0'));
  end;
  close(f);
  While S>=10 do
   S:=Sum_cifr(S);
  assign(f,'korin.eta');rewrite(f);
  WriteLN(f,S);
  close(f);
end.
```

Задача 3. **«Квадроінваріант»**

При піднесенні деяких натуральних чисел у квадрат отриманий результат може закінчуватися даним числом. Наприклад, $76^2 = 5776$. Визначити всі **N**-значні числа, що мають зазначену властивість (**N** ? 20).

Вхідний файл: **VARIANT.DAT N.**

Вихідний файлб **VARIANT.ETA** повинен містити усі числа, що мають задану властивістю, кожне з яких виводиться з нового рядка. Порядок виведення значення не має.

Приклад	*Вхідні дані*	*Вихідні дані*
	VARIANT.DAT	**VARIANT.ETA**
Тест	2	76 25

Пояснення до розв'язань

Ідея розв'язання базується на тому факті, що **n**-цифрове число **a**, піднесене до квадрата, лише тоді даватиме число, останні цифри якого складають саме число **a**, коли число, яке отримується з **a** відкиданням першої цифри, теж має таку властивість.

Цей факт доводиться елементарною підстановкою: нехай $a = \overline{a_1 a_2 \ldots a_n}$ та відомо, що $\overline{a_2 \ldots a_n}^2 = x \cdot 10^{n-1} + \overline{a_2 \ldots a_n}$ (тобто, число

$\overline{a_2...a_n}$, піднесене до квадрату, закінчується числом $\overline{a_2...a_n}$). Тоді маємо:

$$\overline{a_1a_2...a_n}^2 = (a_1 \cdot 10^{n-1} + \overline{a_2...a_n})^2 =$$
$$= a_1^2 \cdot 10^{2n-2} + 2 \cdot a_1 \cdot 10^{n-1} \cdot \overline{a_2...a_n} + \overline{a_2...a_n}^2 =$$
$$= a_1^2 \cdot 10^{2n-2} + 2 \cdot a_1 \cdot 10^{n-1} \cdot \overline{a_2...a_n} + x \cdot 10^{n-1} + \overline{a_2...a_n} =$$
$$= a_1^2 \cdot 10^{2n-2} + \left(2 \cdot a_1 \cdot \overline{a_2...a_n} + x\right) \cdot 10^{n-1} + \overline{a_2...a_n}.$$

У цьому числі останні $n-1$ цифра складають число $\overline{a_2...a_n}$, оскільки у записаній сумі лише останній доданок має ненульові останні $n-1$ цифру.

Виходячи з другого доданка, n-та з кінця цифра результату піднесення до квадрата є останньою цифрою числа $2 \cdot a_1 \cdot \overline{a_2...a_n} + x$, вона має дорівнювати a_1. Звідси однозначно визначається a_1.

Таким чином, числа, які мають описану в умові властивість, довжини n можна одержувати з чисел довжини $n-1$, які мають цю ж властивість.

Розв'язання мовою Паскаль:

```
Program O_02_03_3;
const maxn=20;
var o:text;
    l,n,m,x,t,p,i,j:byte;
    a,b:array[1..2,1..maxn]of byte;
    r:array[1..maxn]of byte;
begin
  assign(o,'variant.dat'); reset(o);
  readln(o,n);  close(o);
  assign(o,'variant.eta'); rewrite(o);
  if n=1
  then begin
    writeln(o,1);
    writeln(o,5);
    writeln(o,6);
  end;
  a[1,maxn-1]:=2; a[1,maxn]:=5;
  a[2,maxn-1]:=7; a[2,maxn]:=6;
  for l:=3 to n do
  begin
    for m:=1 to 2 do
```

```
    begin
      {обчислюємо значення квадрата}
      fillchar(r,sizeof(r),0);
      for i:=maxn downto maxn-l+1 do
      begin
        p:=0;
        for j:=maxn downto maxn-l+1 do
        begin
          t:=r[i+j-maxn]+a[m,i]*a[m,j]+p;
          r[i+j-maxn]:=t mod 10;
          p:=t div 10;
        end;
      end;
      {остання цифра числа X}
      x:=r[maxn-l+1];
      b[m]:=a[m];
      {знаходимо нову цифру a1=b[m,maxn-l+1]}
      b[m,maxn-l+1]:=1;
      while (b[m,maxn-l+1]<10)and
         ((2*b[m,maxn-l+1]*a[m,maxn]+x)
             mod 10 <> b[m,maxn-l+1])
      do inc(b[m,maxn-l+1]);
      {тривіальний випадок: a1=0}
      if b[m,maxn-l+1]=10
      then b[m,maxn-l+1]:=0;
    end;
    a:=b;
  end;
  {виводимо знайдені числа}
  for m:=1 to 2 do
    {тривіальний випадок не виводимо,}
    {оскільки це число не n-значне}
    if a[m,maxn-n+1]<>0
    then begin
      for i:=maxn-n+1 to maxn do
        write(o,a[m,i]);
      writeln(o);
    end;
  close(o);
end.
```

Задача 4. Ювілей

Будемо називати *віковим ювілеєм* великої людини 100-річчя, 200-річчя і т.д. з дня його народження. У місті збираються святкувати віковий ювілей одного знаменитого поета. На честь цього вирішено за рік до ювілею встановити в центрі міста табло, на якому повинно відображатися кількість днів, що залишилося до цієї визначної дати.

Потрібно написати програму для роботи з цим табло, що за датою народження поета і поточною датою визначає кількість днів, що залишилися до дня народження.

Примітка. Програма повинна забезпечити правильну роботу табло з 1999 по 2099 р.р.

Вхідний файл: **UVILEY.DAT** містить два рядки, у першому рядку — дата народження поета у форматі **дд.мм.рррр.**, у другому рядку — поточна дата в тому ж форматі.

Вихідний файл: **UVILEY.ETA** написана вами програма повинна помістити у вихідний файл або кількість днів, або повідомлення «*Свято вже триває*», у випадку, якщо день народження вже настав або минув.

Приклад	*Вхідні дані*	*Вихідні дані*
	UVILEY.DAT	**UVILEY.ETA**
Тест	06.06.1799 01.05.1999	36

Пояснення до розв'язань

Задача є нескладною алгоритмічно, але вимагає уважності для урахування всіх випадків.

Очевидно, що необхідно розглянути такі випадки:

1. Поточний рік є ювілейним (різниця між роками кратна 100).
2. Поточний рік напередодні ювілейного (різниця між роками кратна 99).

У першому варіанті залишається порівняти місяці. Якщо вони однакові, кількість днів обчислюється елементарно (різниця номерів днів). У протилежному випадку обчислюється кількість днів у місяцях, що йдуть між поточним та місяцем народження, до якої додаються залишок днів поточного місяця та кількість днів с початку місяця до дня народження.

У другому випадку необхідно підрахувати кількість днів до кінця поточного року та кількість днів с початку наступного року до дати народження.

Оскільки за умовою задачі табло встановлюється за рік до дня народження, вважається, що переходу більш ніж між двома роками бути не може. Крім того, на кожному етапі перевіряється, чи поточна дата не більша від ювілейної (поточний рік більший від ювілейного, або при рівних роках поточний місяць більший від ювілейного, або при рівних місяцях поточний день більший або дорівнює дню народження. У всіх цих випадках видається повідомлення *«Свято вже триває»*.

Підпрограми у запропонованій програмі виконують такі дії:

- Процедура `Input` — організовує уведення дати та розбиття її на складові: рік, місяць, день.
- Функція `Leap_year` перевіряє, чи є рік, що перевіряється, високосним.
- Функція `month_day` підраховує кількість днів з місяця `month1` до місяця `month2`.
- Функція `Count_day` підраховує кількість днів до кінця поточного місяця з урахуванням високосного року.

```
Program O_02_03_4;
Procedure Input(var f:text;var year,month, day:
word);
var s:string;k:integer;
begin
  readln(f,s);
  Val(Copy(s,1,2),day,k);
  Val(Copy(s,4,2),month,k);
  Val(Copy(s,7,4),year,k);
end;
Function Leap_year(year:word):boolean;
begin
  Leap_year:=(year mod 4=0)and(year mod 400 <>0);
end;
Function month_day(month1,month2,year:word):word;
var i:word; count:word;
begin
  count:=0;
  for i:=month1 to month2 do
    case i of
```

```
        1,3,5,7,8,10,12: count:=count+31;
        4,6,9,11: count:=count+30;
        2: if Leap_year(year)
           then count:=count+29
           else count:=count+28;
     end;
     Month_day:=count
end;
Function Count_day(day,month,year:word):word;
begin
 case month of
   1,3,5,7,8,10,12: count_day:=31-day;
   4,6,9,11: count_day:=30-day;
   2: if Leap_year(year)
      then count_day:=29-day
      else count_day:=28-day;
 end;
end;
var f:text;
year_n,year_p,month_n,month_p,day_n,day_p:word;
year,day:word;
Begin
  assign(f, 'uviley.dat');reset(f);
  input(f,year_n,month_n,day_n);
  input(f,year_p,month_p,day_p);
  close(f);
  assign(f, 'uviley.eta'); rewrite(f);
  if (year_p-year_n) mod 100=0
  then if month_n<month_p
       then writeln(f, 'Свято вже йде')
       else if month_n=month_p
            then if day_n<=day_p
                 then writeln(f, 'Свято вже йде')
                 else writeln(f,day_n-day_p)
            else begin
                 day:=month_day(month_p+1,
                 month_n-1,year_p)+day_n;
            day:=day+count_day(day_p,month_p,year_p);
                 writeln(f,day);
                end
```

```
    else if (year_p-year_n) mod 100=99
         then begin
                day:=month_day(month_p+1,12,year_p)
           +month_day(1,month_n-1,year_p+1)+day_n;
           day:=day+count_day(day_p,month_p,year_p);
               writeln(f,day);
              end
         else writeln(f,'Свято вже йде');
    close(f);
End.
```

10—11 клас

Задача 5. **Обертання слова**

Потрібно написати програму, що повинна надрукувати слово, отримане з вхідного циклічним зсувом його на **N** символів вліво. Будемо називати словом будь-яку послідовність букв латинського алфавіту **A-Z, a-z**. На вході програми задається слово довжини **L** < 80 і натуральне число **N** $< 10^{100}$. При циклічному зсуві на кожному кроці буква слова, що стоїть на першому місці, переміщується в кінець.

Вхідний файл: **SLOVO.DAT** у першому рядку вхідного файла задане число **N**, у другому — вхідне слово.

Вихідний файл: **SLOVO.ETA** містить один рядок з отриманим словом.

Приклад	*Вхідні дані*	*Вихідні дані*
	SLOVO.DAT	**SLOVO.ETA**
Тест	3 Computer	PuterCom

Пояснення до розв'язань

Очевидно, що повне моделювання процесу зсуву в задачі неможливе, оскільки кількість зсувів є довгим числом. Крім того, це і не потрібно, адже зсув на кількість позицій, що дорівнює довжині рядка, повертає його у початковий стан. Виходячи з цього, пропонуємо наступний алгоритм. Знаходимо для числа, що дорівнює кількості зсувів, остачу від ділення на довжину рядка, а вже потім моделюємо процес зсувів, використовуючи особливості роботи з рядками: спочатку перший символ додаємо в кінець рядка, а потім вилучаємо його процедурою **Delete**.

Оскільки, як було сказано, кількість зсувів — довге число, програма містить дві підпрограми для його обробки:

- процедура **Input** — для введення числа з файлу та конвертування його у масив (нульвий елемент масиву містить кількість цифр у числі);
- функція **Division** — знаходить частку від ділення довгого числа на коротке (напівдовге ділення).

```
Program O_02_03_5;
const n=20;
type Long=array[0..n] of byte;
Procedure Input(var f:text; var Number:Long);
var i:byte; s:string;
begin
  readln(f,S);
  fillchar(Number,sizeof(Number),0);
  for i:=length(S) downto 1 do
    Number[n+i-length(S)]:=ord(S[i])-ord('0');
  Number[0]:=length(S);
end;
Function Division(X:Long;k:byte):word;
var i:byte; Rez:Long; ost:word;
begin
  fillchar(rez,sizeof(rez),0);
  ost:=0;
  for i:=1 to X[0] do
  begin
    ost:=ost*10+X[n-X[0]+i];
    Rez[n-X[0]+i]:=ost div k;
    ost:=ost mod k;
  end;
  Division:=ost;
end;
var f:text; i,Numb:word; st1:string; Num:Long;
Begin
  assign(f, 'slovo.dat');reset(f);
  input(f,Num);
  readln(f,st1);close(f);
  Numb:=Division(Num,length(st1));
  for i:=1 to Numb do
  begin
```

```
      st1:=st1+st1[1];
      delete(st1,1,1);
    end;
    assign(f, 'Slovo.eta'); rewrite(f);
    WriteLN(f,st1);
    close(f);
end.
```

Задача 6. Маршрут коника-стрибунця

На лівому кінці лінійки (позначка нуль) довжиною **N** сантиметрів сидить коник-стрибунець, що хоче дістатися її правого кінця (поділка **N**). Коник-стрибунець може робити стрибки тільки визначеної довжини, що не перевищують **N** і тільки вперед.

Потрібно написати програму, що підраховує кількість різних маршрутів коника-стрибунця з початкового положення в кінцеве.

Примітка. Довжина лінійки **N** ≤ 30; кількість різноманітних стрибків **M** ≤ 10.

Вхідний файл: **KONIC.DAT** перший рядок — числа **N** і **M**, розділені пропуском. другий рядок — **M** натуральних чисел, розділених пропуском, кожне з яких є довжиною можливого стрибка. Жодне з чисел не перевищує **N**.

Вихідний файл: **KONIC.ETA** єдиний рядок містить кількість різних маршрутів **S**.

Приклад	*Вхідні дані*	*Вихідні дані*
	KONIC.DAT	**KONIC.ETA**
Тест	3 2 1 2	3

Пояснення до розв'язань

Дивись журнал «Комп'ютер у школі та сімї», № 1, 2004 р.

Задача 7. Вірус

Колонія клітин являє собою квадратну матрицю порядку **N** (**N** < 500). У колонію проникає **M** (**M** < 11) вірусів, що уражають клітини з координатами (X_1, Y_1), ... (X_m, Y_m). За одну

одиницю часу вірус проникає в клітини, сусідні з ураженими (сусідніми вважаються клітини, що мають спільну сторону).

Потрібно написати програму, що визначить час ушкодження всієї колонії.

Вхідний файл: **VIRUS.DAT**

1 рядок — **N**

2 рядок -**M**

3 рядок — X_1, Y_1

4 рядок — X_2, Y_2

M + 2 рядок — X_m, Y_m

Вихідний файл: **VIRUS.ETA** рядок, що містить одне число — час зараження.

Приклад	*Вхідні дані*	*Вихідні-дані*
	VIRUS.DAT	**VIRUS.ETA**
Тест	5 2 1 2 5 5	4

Пояснення до розв'язань

Основною проблемою у цій задачі є створеня масиву необхідного розміру. Оскільки мова програмування Паскаль не дозволяє створювати структури розміром, що перевищує 64 кБ, необхідно використовувати додаткову пам'ять з кучі, застосовуючи для цього вказівники. Зараз на олімпіадах усіх рівнів дозволяється використовувати інші мови програмування, зокрема FreePascal. Ця мова синтаксично не відрізняється від мови Паскаль, але має можливість використовувати більші за розміром структури. Тоді алгоритм реалізується досить нескладно.

Спочатку на місця ураження вірусом у таблиці записуємо одиниці. А потім запускаємо цикл, який працюватиме, доки всі клітинки не будуть відвідані: ознакою цього буде відсутність у таблиці клітинок, що містять нульові значення. При кожному перегляді таблиці шукаємо клітинки з поточним значенням кроку (змінна **k**) і перевіряємо сусідні з нею (по горизонталі, вертикалі, ліворуч та праворуч). Якщо зустрінуться нульові, заповнюємо їх значенням на одиницю більшим від поточного кроку. Щоб на границі масиву не було виходу за його межі, додаємо до масиву додаткові клітинки з усіх боків, так звані «бар'єрні» елементи.

```
Program O_02_03_7;;
var n,m,i,j,k:integer;
              x:boolean;
   a:array[0..501,0..501] of byte;
   f:text;
begin
  fillchar(a,sizeof(a),0);
  assign(f,'virus.dat');
  reset(f);readln(f,n,m);
  for i:=1 to m do
  begin
    read(f,j,k);a[j,k]:=1;
  end;
  close(f);
  x:=true;k:=1;
  while x do
  begin
   x:=false;
   for i:=1 to n do
    for j:=1 to n do
     if a[i,j]=k
     then begin
           if a[i-1,j]=0
           then begin a[i-1,j]:=k+1;x:=true;end;
           if a[i+1,j]=0
           then begin a[i+1,j]:=k+1;x:=true;end;
           if a[i,j-1]=0
           then begin a[i,j-1]:=k+1;x:=true;end;
           if a[i,j+1]=0
           then begin a[i,j+1]:=k+1;x:=true;end;
          end;
    inc(k);
  end;
  assign(f,'virus.sol');
  rewrite(f);
  write(f,k-2);
  close(f);
end.
```

ОЛІМПІАДА 2003 / 2004

8—9 клас

Задача 1. Дільники добутку

Задано **N** натуральних чисел $a_1, a_2, \ldots, a_N$ ($1 \le N \le 20$), кожне з яких знаходиться в інтервалі від 1 до 10000. Написати програму, що обчислює кількість натуральних дільників добутку чисел $a_1, a_2 \ldots a_N$.

***Вхідний файл*: DIVIDERS.DAT** містить у першому рядку число $1 \le N \le 20$ — кількість натуральних чисел. У другому рядку **N** чисел, розділених проміжками.

***Вихідний файл*: DIVIDERS.SOL** містить одне ціле число — кількість натуральних дільників вищезгаданого числа.

Приклад	*Вхідні дані*	*Вихідні дані*
	DIVIDERS.DAT	**DIVIDERS.SOL**
	4 2 3 4 5	16

Пояснення до розв'язань

Загальна кількість дільників добутку всіх чисел дорівнює добутку збільшених на 1 показників степенів загального розкладу даного добутку:

$$a_1 \cdot a_1 \cdot a_1 \cdot \ldots \cdot a_n = 2^{k_1} \cdot 3^{k_2} \cdot 5^{k_3} \cdot \ldots, \quad kil = (k_1+1) \cdot (k_2+1) \cdot (k_3+1) \cdot \ldots$$

Для розв'язання задачі спочатку створюємо таблицю простих дільників (процедура **Symple_divisor**). Оскільки серед перших 10000 натуральних чисел є 1230 простих, масив **Divisor** резервується саме на таку кількість елементів. Масив **Count** має такий самий розмір, оскільки буде містити кількість знайдених простих дільників у розкладі чисел, що уводяться.

В основній програмі читаємо поступово з файла числа і підраховуємо в таблиці **count** кількість відповідних простих дільників, тобто показник їх степеня в розкладі числа. Результат потім обчислюємо згідно з наведеною формулою.

```
Program O_03_04_1;
arr=array[1..1230] of word;
procedure symp_divisor(var d:arr);
```

```
var i,j:word;
  function symple(a:word):boolean;
  var k:word;
  begin
   if a=2 then symple:=true
   else begin
         if (a mod 2 =0)or(a<2)
         then simple:=false
         else begin
               k:=3;
               while (a mod k<>0)and(k<=sqrt(a))
                do k:=k+2;
               if a mod k=0 then simple:=false
               else simple:=true;
              end;
  end;
begin
 j:=1;
 for i:=2 to 10000 do
  if symple(i)
  then begin
         d[j]:=i;inc(j);
       end;
end;
var n,a,i,j:integer;kil:longint;
count,divisor:arr;
f:text;
begin
 symp_divisor(divisor);
 fillchar(count,sizeof(count),0);
 assign(f,'dividers.dat');
 reset(f); readln(f,n);
 for i:=1 to n do
 begin
  read(f,a);
  j:=1;
  while a<>1 do
   if a mod divisor[j]=0
   then begin
         inc(count[j]); a:=a div divisor[j];
```

```
        end
    else inc(j);
  end;
  close(f);
  kil:=1;
  for i:=1 to 1230 do kil:=kil*(count[i]+1);
  assign(f,'divisers.sol');
  rewrite(f);
  write(f,kil);
  close(f);
end.
```

Задача 2. Подвійні числа

Зайшовши до кімнати Незнайка, Знайко застав того за дивним заняттям: він записував число, наприклад, 512, потім на папері щось ділив, виписував, переписував і, врешті, біля першого з'явилося інше число 1. Далі Незнайко записав 513, виконав дії, подібні до попередніх, і поряд з'явилося число 513. Аналогічно були отримані й інші два числа 514 та 257. Зрозумівши ідею записів, Знайко похвалив товариша, але зауважив, що, маючи комп'ютер, не слід користуватися дідівським способом обчислень.

Друзі, допоможіть Незнайці. Напишіть програму, яка б за уведеним цілим додатнім числом виводила зв'язане відповідним чином ціле додатне число.

Вхідні дані: у текстовому файлі **NUMBER.DAT** міститься одне додатне число **N** ($512 \le N \le 1024$).

Вихідні дані: у текстовому файлі **NUMBER.SOL** міститься одне додатне число **M** ($M > 0$).

Приклад	*Вхідні дані*	*Вихідні дані*
	NUMBER.DAT	**NUMBER.SOL**
	512 513 514 1023	1 513 257 1023

Пояснення до розв'язань

Проаналізувавши умову, бачимо, що Незнайко спочатку переводив числа у двійкову систему числення, потім розвертав їх і

переводив назад у десяткову. На наш погляд, цей процес можуть промоделювати всі учні, які володіють непоганою технікою програмування.

Зверніть увагу, що оскільки при переведенні числа у двійкову систему числення кількість цифр може перевищити 10 (максимально можлива у типі **longint**), у наведеному розв'язанні двійкове число зберігається у рядку. Це дозволяє не перевертати число безпосередньо, а просто при зворотному переведенні числа у десяткову систему числення цифри брати зліва направо.

```
Program O_03_04_2;
var n,m,L,i:word;s:string;f:text;
begin
 assign(f, 'number.dat');
 reset(f);readln(f,n);
 close(f);
 s:=' ';
 while n>0 do
 begin
  s:=chr(n mod 2+ord('0'))+s;
  n:=n div 2;
 end;
 m:=0;L:=1;
 for i:=1 to length(s) do
 begin
   m:=m+(ord(s[i])-ord('0'))*L;
   L:=L*2;
 end;
 assign(f,'number.sol');
 rewrite(f);write(f,m);
 close(f);
end.
```

Задача 3. **Нам достатньо півсклянки**

Група друзів зібралися в понеділок і на всі свої гроші купила пляшки з напоєм, не забувши взяти здачу. У вівторок друзі здали порожній посуд, додали здачу, і знову купили стільки напою, скільки могли. Так вони діяли до п'ятниці. У п'ятницю, здавши

посуд і додавши здачу за четвер, вони змогли купити тільки одну пляшку напою.

Потрібно визначити мінімальну суму, яка була в друзів у понеділок.

Технічні зауваження:

- вартість пляшки з напоєм і порожньої пляшки виражаються цілими додатними числами і не перевершують 1 000 000 коп. кожна;
- порожня пляшка дешевша від пляшки з напоєм;
- початкова сума не перевищує 2 000 000 000 коп.

Вхідні дані: у першому рядку файла **HALF.DAT** міститься одне додатне число — вартість пляшки з напоєм, у другому рядку — вартість порожньої пляшки.

Вихідні дані: файл **HALF.SOL** містить одне додатне число — мінімальну суму, яка була в друзів у понеділок.

Приклад	*Вхідні дані*	*Вихідні дані*
	HALF.DAT	**HALF.SOL**
	7 3	83

Пояснення до розв'язань

Дану задачу теж розв'язуємо за допомогою моделювання ситуації. Для цього використовуємо три масиви:

Count — кількість куплених пляшок;

Ost — здача, яку отримали товариші;

Sum — загальна сума грошей, яка потрібна для купівлі.

За умовою здачі у п'ятницю не залишилось і товариші змогли купити лише одну пляшку напою. Тому масив **Count** містить одиницю, масив **Sum** — значення **a** (вартість однієї пляшки), а массив **Ost** — нуль. Далі починаємо підбирати кількість пляшок напою, яку купили у попередній день. При цьому враховуємо, що всі пляшки повинні бути зданими наступного дня. Тобто сума наступного дня разом з остачею (яка, очевидно, не може перевищувати вартості пляшки з напоєм) повинна бути повністю витраченою.

```
Program O_03_04_3;
var a,b,i:longint;f:text;
 count,sum,ost:array[1..5] of word;
begin
  assign(f, 'half.dat');
```

```
  reset(f);read(f,a,b);
  close(f);
  fillchar(count,sizeof(count),0);
  count[5]:=1;ost[5]:=0;sum[5]:=a;
  for i:=4 downto 1 do
  begin
    repeat
      inc(count[i]);
      ost[i]:=sum[i+1]-count[i]*b;
    until ost[i]<a;
    sum[i]:=count[i]*a+ost[i];
  end;
  assign(f, 'half.sol');
  rewrite(f);write(f,sum[1]);
  close(f);
end.
```

Задача 4. **Оптимальний маршрут**

Є смужка з клітинками, пронумерованими від 1 до 60. В одній з них знаходиться виконавець, що вміє виконувати команди:

1) крок уперед (перейти на сусідню клітинку з більшим номером);

2) крок назад (перейти на сусідню клітинку з меншим номером);

3) стрибок уперед (перемістити в клітинку, номер якої більше номера поточної клітинки на 8);

4) стрибок назад (перемістити в клітинку, номер якої менше номера поточної клітинки на 8);

5) у початок (стати на клітинку номер 1);

6) у кінець (стати на клітинку номер 60).

Якщо виконання команди приведе до виходу за межі смужки, виконавець її ігнорує.

Написати програму, що за уведеним L (номером клітинки з початковим положенням виконавця) і M (номером клітинки, куди потрібно потрапити) знаходить алгоритм руху з найменшою кількістю команд. Результат вивести у вигляді: команда, номер поля після команди.

Технічна вимога: при виведенні алгоритму руху спочатку розташувати команди стрибків, а потім — кроків.

Примітка. Для деяких **L** і **M** існує кілька оптимальних розв'язань.

Вхідні дані: у першому рядку файла **OPTIM.DAT** міститься одне додатне число — номер клітинки з початковим положенням виконавця, у другому рядку — номер клітинки, куди потрібно потрапити.

Вихідні дані: файл **OPTIM.SOL** містить рядки, в кожному з яких вказано: команду, номер поля після команди.

Приклад	*Вхідні дані*	*Вихідні дані*
	OPTIM.DAT	**OPTIM.SOL**
	44 11	Оптимальний маршрут: у початок 1 стрибок уперед 9 крок уперед 10 крок уперед 11

Пояснення до розв'язань

Використаємо метод пошуку в ширину. Таблиця **a** використовується для позначення відвіданих клітинок; таблиця **b** — для позначення способу попадання у дану клітинку.

```
Program O_03_04_4;
 var f: text; L,m,i,j,k: byte;
 a,b: array[1..60]of integer;
 c:array[1..60]of string;
 {c - допоміжна таблиця для формування запису
      маршруту}
begin
 assign(f, 'optim.dat');
 reset(f);read(f,L,m);
 close(f);
 for i:=1 to 60 do begin a[i]:=0;b[i]:=0;end;
 a[L]:=1;j:=1;
 if L<>1 then begin a[1]:=1;b[1]:=-2;end;
 if L<>60 then begin a[60]:=1;b[60]:=2;end;
 while b[m]=0 do
 begin
   inc(j);
   for i:=1 to 60 do
   begin
```

```
      if (i-1>0)and(a[i-1]=0)and(a[i]=j-1)
      then begin a[i-1]:=j;b[i-1]:=1;end;
      if (i+1<61)and(a[i+1]=0)and(a[i]=j-1)
      then begin a[i+1]:=j;b[i+1]:=-1;end;
      if (i-8>0)and(a[i-8]=0)and(a[i]=j-1)
      then begin a[i-8]:=j;b[i-8]:=8;end;
      if (i+8<61)and(a[i+8]=0)and(a[i]=j-1)
      then begin a[i+8]:=j;b[i+8]:=-8;end;
    end;
  end;
  k:=m;
  for i:=j downto 1 do
    case b[k] of
     8:begin c[i]:='Prizok nazad'+
         chr(48+k div 10)+chr(48+k mod 10);
         k:=k+b[k];
       end;
     2:begin c[i]:='U pochatok'+
         chr(48+k div 10)+chr(48+k mod 10);
         k:=k+b[k];
       end;
     1:begin c[i]:='Krok nazad'+
         chr(48+k div 10)+chr(48+k mod 10);
         k:=k+b[k];
       end;
    -1:begin c[i]:='Krok upered'+
         chr(48+k div 10)+chr(48+k mod 10);
         k:=k+b[k];
       end;
    -2:begin c[i]:='U kinets'+
         chr(48+k div 10)+chr(48+k mod 10);
         k:=k+b[k];
       end;
    -8:begin c[i]:='Prizok upered'+
         chr(48+k div 10)+chr(48+k mod 10);
         k:=k+b[k];
       end;
    end;
  assign(f, 'optim.sol');
  rewrite(f);writeln(f, 'Optimum:');
```

```
  for i:=1 to j do
    writeln(f,c[i]);
  close(f);
end.
```

10—11 клас

Задача 5. **Дитяче свято**

Організатори дитячого свята планують надути для нього **M** повітряних кульок. З цією метою вони запросили **N** добровільних помічників, **i**-й серед яких надуває кульку за T_i хвилин, проте кожний раз після надування Z_i кульок стомлюється і відпочиває Y_i хвилин. Тепер організатори свята хочуть дізнатися, через який час будуть надуті всі кульки при найоптимальнішій роботі помічників і скільки кульок надує кожен з них. (Якщо помічник надув кульку й повинен відпочити, але більше кульок йому надувати не доведеться, то вважається, що він закінчив роботу одразу після закінчення надування останньої кульки, а не після відпочинку).

Вхідні дані: у текстовому файлі **CHILDREN.DAT** у першому рядку знаходяться числа **M** та **N** ($0 \le M \le 1000, 1 \le N \le 20$), які розділені проміжком. Наступні **N** рядків містять по три числа — T_j, Z_j, і Y_j відповідно ($1 \le T_j, Y_j \le 100, 1 \le Z_j \le 1000$).

Вихідні дані: у текстовому файлі **CHILDREN.SOL** у першому рядку виведіть число **T** — час, за який будуть надуті всі кульки. У другому рядку виведіть **N** чисел — скільки кульок надує кожний із запрошених помічників. Розділяйте числа проміжками. Якщо розподілів кілька, виведіть будь-який із них.

Приклад	*Вхідні дані*	*Вихідні дані*
	CHILDREN.DAT	**CHILDREN.SOL**
	10 3 1 2 3 3 10 3 2 4 3	8 4 2 4
	1 3 1 1 100 2 1 100 3 1 100	1 1 0 0

Пояснення до розв'язань

Задача розв'язується за принципом динамічного програмування.

Позначимо **F(n,m)** мінімальний час, за який **n** помічників надують **m** кульок. Зрозуміло, що за наявності лише одного помічника він сам має надути усі кульки, тому час його роботи дорівнюватиме $T_1 \cdot m + Y_1 \cdot ((m-1) \text{ div } Z_1)$.

Нехай ми підрахували значення **F** за умови використання перших **n** помічників. Якщо ми залучимо до роботи ще одного додаткового помічника (з номером **n+1**), то він, можливо, зможе зменшити загальний час роботи, взявши частину роботи на себе — надувши від 0 до **m** кульок. На надування k кульок він витратить час $T_{n+1} \cdot k + Y_{n+1} \cdot ((k-1) \text{ div } Z_{n+1})$. Оскільки інші помічники працюють незалежно від нього, час їхньої роботи можна оцінити значенням $F(n,m-k)$. Тобто загальний час роботи: $\max\{F(n,m-k),\ T_{n+1} \cdot k + Y_{n+1} \cdot ((k-1) \text{ div } Z_{n+1})\}$. З усіх можливих значень k слід обрати таке, на якому досягається мінімум записаної величини. Відповіддю першої частини запитання, поставленого у задачі, буде значення **F (n,m)**.

Для знаходження кількості кульок, які має надути кожен помічник, щоб досягти знайденого мінімального часу, слід для кожного значення **F (n,m)** запам'ятовувати те значення **k**, на якому досягається мінімум часу: це і буде оптимальна кількість кульок, які має взяти **n**-й помічник.

Розв'язання мовою Паскаль:

```
Program O_03_04_5;
const maxn = 20;
      maxm = 100;
type  balls = 0..maxm;
      people = 0..maxn;
      time = longint;
var i, n: people;
    j, k, m: balls;
    max: time;
    T, Z, Y: array [people] of time;
    F: array [people,balls] of time;
    Take: array [people,balls] of balls;
```

```
    optimum: array [people] of balls;
    fi:text;
begin
  assign(fi, 'children.dat');
  reset(fi);
  read(fi,m,n);
  for i:=1 to n do
    read(fi,T[i],Z[i],Y[i]);
  close(fi);
  fillchar(F,sizeof(F),0);
  for j:=1 to m do
  begin
    Take[1,j]:=j;
    F[1,j]:=T[1]*j+Y[1]*((j-1) div Z[1]);
  end;
  for i:=2 to n do
    for j:=1 to m do
    begin
      Take[i,j]:=0;
      F[i,j]:=F[i-1,j];
      for k:=1 to j do
      begin
        max:=T[i]*k+Y[i]*((k-1) div Z[i]);
        if F[i-1,j-k]>max then max:=F[i-1,j-k];
        if max<F[i,j]
        then begin
               F[i,j]:=max;
               Take[i,j]:=k;
             end;
      end;
    end;
  assign(fi, 'children.sol');
  rewrite(fi);
  writeln(fi,F[n,m]);
  for i:=n downto 1 do
  begin
    optimum[i]:=Take[i,m];
    m:=m-Take[i,m];
  end;
  for i:=1 to n-1 do
```

```
    write(fi,optimum[i],' ');
  writeln(fi,optimum[n]);
  close(fi);
end.
```

Задача 6. Скарбничка

Для того щоб почати бізнес, юний комерсант вирішив накопичити кошти. З цією метою він відшукав скарбничку і почав збирати гроші.

Відомо, що визначити накопичену суму в скарбничці можна тільки розбивши цю скарбничку. Однак юному комерсанту не хотілося робити це завчасно, тобто доки там не назбиралася необхідна сума. Уникнути цього йому допоміг його напарник, що порадив, як можна оцінити мінімальну кількість грошей усередині скарбнички, знаючи її вагу без монет, вагу з монетами і вагу монет кожного типу.

Потрібно написати програму, яка визначала б мінімальну суму грошей, що може знаходитися в скарбничці, за відомими вхідними даними.

Вхідні дані: файл **MONEY.DAT** містить у першому рядку два цілих числа:

E — вагу порожньої скарбнички ($1 \le E \le 10000$);

F — вагу скарбнички, заповненої монетами ($1 \le E \le F \le 10000$). У другому рядку міститься ціле число **N** ($1 \le N \le 500$) — кількість типів монет.

Далі — **N** рядків опису монет різних типів. Кожен рядок містить два цілих числа — P_i, і W_i ($1 \le P_I \le 50000$, $1 \le W_I \le 10000$, $1 \le I \le N$, де P_i — номінал монети **i**-го типу, а W_i — її вага.

Вихідні дані: у файл **MONEY.SOL** записати одне ціле число — мінімальну суму грошей, що може знаходитись у скарбничці. Якщо задана вага скарбнички **F** не може бути отримана монетами заданих типів, вихідний файл повинен містити повідомлення **"NO SOLUTION"**.

Пояснення до розв'язань

Ця задача є різновидом класичної «задачі про наплічник».

Введемо функцію $Q(n,m)$, яка позначатиме найменшу кількість грошей, які можуть бути у скарбниці, якщо маса нетто

монет становить m і в наявності є монети лише перших n номіналів. Одразу домовимось, що якщо набрати задану масу за допомогою монет, що є у наявності, неможливо, то $Q(n,m) = +\infty$. Зрозуміло, що якщо у наявності є лише монети першого номіналу, то задану масу можна отримати лише за їх допомогою — або не отримати взагалі. Виходячи з цього, запишемо:

$$Q(1,m) = \begin{cases} \left(m / w_1\right) \cdot p_1, & \text{якщо } m \text{ ділиться націло на } w_1 \\ +\infty, & \text{інакше} \end{cases}$$

Тепер нехай ми вже знаємо значення $Q(i,j)$ для $i = \overline{1,n-1}$, $\forall j$. Додамо до розгляду монети ще одного номіналу й підрахуємо значення $Q(n,j)$:

Зрозуміло, що на m одиниць маси може припадає від 0 до $t = \left[m / w_n\right]$ монет n-го номіналу. Якщо припустити, що на m одиниць маси припадає k монет n-го номіналу, то зрозуміло, що найменша можлива загальна вартість монет складає $R_{nk} = p_n \cdot k + Q(n-1, m - w_n \cdot k)$. Серед усіх варіантів значення $k \in [0,t]$ слід обрати таке, на якому досягається мінімум серед R_{nk}. Остаточною відповіддю на запитання задачі є значення $Q(N, F-E)$ (оскільки маса нетто монет, які є у скарбниці, дорівнює $(F-E)$).

Зауважимо наостанок, що, оскільки значення $Q(i,j)$ обчислюється через значення $Q(i-1, j-k)$, можна не створювати прямокутної таблиці розмірами $N \times (F-E)$. Натомість можна зберігати значення $Q(i-1)$ у $Q(0)$, а значення $Q(i)$, яке обчислюється, — у $Q(1)$ і динамічно виконувати переприсвоєння. Окрім того, можна помітити, що значення $Q(i,j)$ завжди оптимальніше (або таке саме), ніж значення $Q(i-1,j)$, тому можна при подальших

обчисленнях вільно заміщувати $Q(i-1,j)$ на $Q(i,j)$. З цих міркувань випливає, що можна взагалі обійтися лінійним масивом Q: якщо у цьому масиві на початку n-го кроку зберігалися значення $Q(n-1,j)$, то нові значення обчислюватимуться послідовно, заміщуючи старі, за принципом $Q(i)=\min\limits_{k\in[0..t_n]}\{p_n\cdot k+Q(i-k\cdot w_n)\}$ (де $t_n=\left[i/w_n\right]$).

Розв'язання мовою Паскаль:

```
Program O_03_04_6;
const maxn = 500;
      maxm = 10000;
      infinite = 1000000;
var E, F, n, i, j: integer;
    sum: longint;
    P, W: array [1..maxn] of longint;
    Q: array [1..maxm] of longint;
    fi:text;
begin
  assign(fi, 'money.dat');
  reset(fi);
  read(fi,E,F,n);
  for i:=1 to n do read(fi,P[i],W[i]);
  close(fi);
  sum:=F-E;
  for j:=1 to sum do
    if j mod W[1]=0
    then Q[j]:=(j div W[1])*P[1]
    else Q[j]:=infinite;
  for i:=2 to n do
    for j:=1 to sum do
      for k:=1 to j div W[i] do
        if P[i]*k+Q[j-W[i]]*k<Q[j]
        then Q[j]:=P[i]*k+Q[j-W[i]]*k;
  assign(fi, 'money.sol');
  rewrite(fi);
```

```
    if Q[1,sum]<infinite
    then writeln(fi,Q[1,sum])
    else writeln(fi,'NO SOLUTION');
    close(fi);
end.
```

Задача 7. **Сірники**

На стіл кинули багато сірників, які мають різну довжину. Необхідно визначити, чи з'єднані різні пари сірників. Задано координати кінців усіх сірників. Співпадіння кінців також уважається з'єднанням. Два сірники можуть мати непрямий зв'язок, якщо вони з'єднані через інші з'єднані сірники.

Вихідні дані: для кожної пари **a** та **b** необхідно написати у файл **MATCHES . SOL** : «**CONNECTED**», якщо **a** з'єднано з **b**, та «**NOT CONNECTED**» у протилежному випадку.

Приклад	*Вхідні дані*	*Вихідні дані*
	MATCHES . DAT	**MATCHES . SOL**
	2 2 1 6 5 8 4 14 2 1 2	NOT CONNECTED
	2 2 1 6 5 4 6 12 2 1 2	CONNECTED

Пояснення до розв'язань

Ця задача поєднує у собі просту геометрію та елементарну теорію графів. Спочатку побудуємо для сірників матрицю їх суміжності M, у якій зазначимо, які сірники перетинаються безпосередньо. Для визначення того факту, чи перетинаються два сірники, перевіримо, чи обидва кінці першого з них знаходяться по різні боки від прямої, заданої другим, і навпаки.

Після формування матриці суміжності слід запустити на графі пошук у глибину, який позначить сірники кожної компоненти зв'язності унікальним номером. Далі пара сірників зв'язана, якщо вони обидва знаходяться в одній і тій самій компоненті зв'язності.

Розв'язання мовою Паскаль:

```
Program O_03_04_7;
const maxn = 100;
var i,j,k,n:byte;
    x1,y1,x2,y2,c:array [1..maxn] of integer;
function con(i,j:byte):boolean;
begin
  con:=(((x1[i]-x1[j])*(y2[j]-y1[j]) -
         (x2[j]-x1[j])*(y1[i]-y1[j])) *
        ((x2[i]-x1[j])*(y2[j]-y1[j]) -
         (x2[j]-x1[j])*(y2[i]-y1[j]))<=0) and
       (((x1[j]-x1[i])*(y2[i]-y1[i]) -
         (x2[i]-x1[i])*(y1[j]-y1[i])) *
        ((x2[j]-x1[i])*(y2[i]-y1[i]) -
         (x2[i]-x1[i])*(y2[j]-y1[i]))<=0);
end;
procedure dfs(x:byte);
var i:byte;
begin
  c[x]:=k;
  for i:=1 to n do
    if (c[i]=0) and (con(x,i))
    then  dfs(i);
end;
var fi:text;
begin
  assign(fi, 'matches.dat');
  reset(fi);
  read(fi,n);
  for i:=1 to n do
    read(fi,x1[i],y1[i],x2[i],y2[i]);
  close(fi);
  for i:=1 to n do c[i]:=0;
  k:=0;
  for i:=1 to n do
    if (c[i]=0)
    then begin
           inc(k);
           dfs(i);
         end;
```

```
    assign(fi, 'matches.sol');
    reset(fi);
    for i:=1 to n do
      for j:=i+1 to n do
        if (c[i]=c[j])
        then  writeln(fi, 'CONNECTED')
        else  writeln(fi, 'NOT CONNECTED');
    close(fi);
end.
```

Задача 8. **Скільки шматочків**

На площині задано **N** прямих. Напишіть програму, що визначає, на скільки частин ці прямі розбивають площину.

Вхідні дані: файл **PART.DAT** містить у першому рядку $1<N \le 100$ — кількість прямих. Далі **N** рядків, кожен з яких містить чотири дійсних числа a_i, b_i, c_i, d_i (пари чисел $(a_i b_i)$, (c_i, d_i) — дійсні координати двох різних точок, через які проходить пряма з номером **i**).

Вихідні дані: необхідно написати у файл **PART.SOL** кількість частин, на які розбивають ці прямі площину.

Приклад	*Вхідні дані*	*Вихідні дані*
	PART.DAT	**PART.SOL**
	3 0 0 0 2 0 2 2 0 2 0 0 0	7
	3 -2 -2 2 2 0 -2 4 2 2 -2 -2 2	6

Пояснення до розв'язань

Зауважимо, що всі фігури, на які площина розбивається прямими, являють собою опуклі області — обмежені або необмежені.

Будемо розбивати площину поступово — спочатку розіб'ємо її лише першою з прямих, далі додаватимемо чергову пряму, розбиваючи шматки площини, одержані на попередніх кроках, на додаткові частини.

Спробуймо знайти закономірність у кількості частин площини, яка отримується, якщо ми маємо у своєму розпорядженні лише три прямі (рис. 1).

Рис. 1. Випадки розбиття для двох прямих

Проводячи одну пряму, ми в будь-якому разі розбиваємо площину на дві півплощини. Коли у розгляді дві прямих вони можуть або співпадати, або перетинатися в одній точці, або бути паралельними. Вочевидь, кількість частин, на яку ці прямі розбивають площину, дорівнює відповідно 2, 4 або 3.

На рис. 2 зображено різні випадки розміщення третьої прямої щодо перших двох (випадки, коли третя пряма співпадає з однією з перших двох, не будемо розглядати окремо).

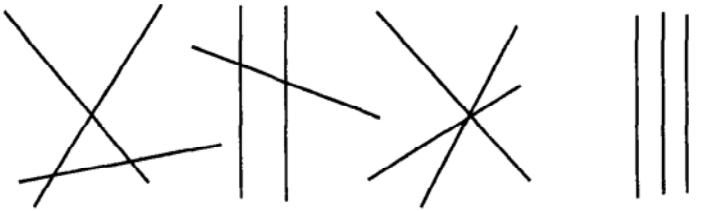

Рис. 2. Випадки розбиття для трьох прямих

Виходячи з цього, можна помітити закономірність: кількість частин, які одержуються після додавання чергової прямої, відрізняється від кількості частин площини, які були до цього, рівно на число, на одиницю більше за кількість перетинів новододаної прямої із тими прямими, які вже були. При цьому, якщо в одній точці перетинаються більше двох прямих, це враховується як один перетин.

Математичний апарат

Рівняння прямої, що проходить через дві точки:

$$\frac{x-x_1}{y-y_1}=\frac{x_2-x_1}{y_2-y_1}.$$

Знайдемо точку перетину двох прямих із системи:

$$\begin{cases} \dfrac{x-x_{11}}{y-y_{11}}=\dfrac{x_{12}-x_{11}}{y_{12}-y_{11}} \\ \dfrac{x-x_{21}}{y-y_{21}}=\dfrac{x_{22}-x_{21}}{y_{22}-y_{21}} \end{cases} \Leftrightarrow \begin{cases} (x-x_{11})(y_{12}-y_{11})=(x_{12}-x_{11})(y-y_{11}) \\ (x-x_{21})(y_{22}-y_{21})=(x_{22}-x_{21})(y-y_{21}) \end{cases}$$

$$\begin{cases} (y_{12}-y_{11})x+(x_{11}-x_{12})y+(x_{12}y_{11}-x_{11}y_{12})=0 \\ (y_{22}-y_{21})x+(x_{21}-x_{22})y+(x_{22}y_{21}-x_{21}y_{22})=0 \end{cases} \Leftrightarrow \begin{cases} a_1x+b_1y+c_1=0 \\ a_2x+b_2y+c_2=0 \end{cases}$$

З другого рівняння $y=-\dfrac{a_2}{b_2}x-\dfrac{c_2}{b_2}$. Підставимо цей вираз у перше рівняння:

$$a_1x+b_1\left(-\frac{a_2}{b_2}x-\frac{c_2}{b_2}\right)+c_1=0,$$

$$x=\frac{b_1c_2-b_2c_1}{a_1b_2-a_2b_1} \Rightarrow y=\frac{a_2}{b_2}\frac{c_1b_2-c_2b_1}{a_1b_2-a_2b_1}-\frac{c_2}{b_2}.$$

Розв'язання мовою Паскаль:

```
Program O_03_04_8;
const maxn = 100;
      eps = 1e-4;
var cnt, i, j, k, n: byte;
 x1,y1,x2,y2,a,b,c,ix,iy: array [1..maxn] of real;
    xin, yin: real;
    num: word;
    f:text;
begin
  assign(f, 'part.dat');
  reset(f);
  read(f,n);
  for i:=1 to n do
  begin
    read(f,x1[i],y1[i],x2[i],y2[i]);
    a[i]:=y2[i]-y1[i];
    b[i]:=x1[i]-x2[i];
    c[i]:=x2[i]*y1[i]-x1[i]*y2[i];
  end;
```

```
  close(f);
  num:=2;
  for i:=2 to n do
  begin
    cnt:=0;
    j:=1;
    { чи не співпадає ця пряма з якоюсь }
   while (j<i)and((abs(a[i]*b[j]-a[j]*b[i])>eps)
       or(abs(c[i]*b[j]-c[j]*b[i])>eps))
    do  inc( j );
    if (j=i)
    then begin
          for j:=1 to (i-1) do
            if abs(a[i]*b[j]-a[j]*b[i])>eps
            then begin
                  xin:=(b[i]*c[j]-b[j]*c[i])/
                       (a[i]*b[j] - a[j]*b[i]);
                  if abs(b[j])<eps
                 then yin:=-(a[i]*xin+c[i])/b[i]
                else yin:=-(a[j]*xin+c[j])/b[j];
                   { чи не було вже цьої точки }
                   k:=1;
                   while (k<=cnt) and
                    ((abs(xin-ix[k])>eps)
                    or (abs(yin-iy[k])>eps))
                   do inc(k);
                   if (k>cnt)
                   then begin
                          inc(cnt);
                          ix[cnt]:=xin;
                          iy[cnt]:=yin;
                         end;
                  end;
           num:=num+cnt+1;
         end;
  end;
  assign(f, 'part.sol');
  rewrite(f);  writeln(f,num);
  close(f);
end.
```

ОЛІМПІАДА 2004 / 2005

Задача 1. Обіцянки мера

Щоб розв'язати проблему безробіття, мер міста **N**-ська вирішив побудувати додатково критий ринок в центі міста. Мер знає, що чим більший ринок він побудує, тим більше безробітних можна буде працевлаштувати. Допоможіть меру знайти якомога більшу прямокутну ділянку в центрі міста. Інформація про центральну частину міста зберігається у вигляді таблиці.

Вхідні дані: у текстовому файлі **MARKET.DAT** перший рядок містить два додатні цілі числа **N** (кількість рядків таблиці) та **M** (кількість стовпчиків таблиці) через пробіл ($2 \le N \le 20$, $2 \le M \le 20$). У наступних рядках записані значення таблиці: 0 та 1. Причому 0 — вільна ділянка, а 1 — забудована ділянка.

Вихідні дані: у текстовому файлі **MARKET.SOL** у першому рядку повинна міститись максимальна площа вільної ділянки, у другому рядку — через пробіл йдуть номер рядка та стовпчика лівого верхнього кута прямокутної ділянки, а третій рядок містить номер рядка і стовпця правого нижнього кута (також через пробіл). Якщо вільної ділянки немає, у вихідний файл вивести 0.

MARKET.DAT	MARKET.SOL
3 4 1 0 0 0 0 0 0 0 1 0 0 0	9 1 2 3 4
3 4 1 1 1 1 1 1 1 1 1 1 1 1	0

Пояснення до розв'язань

Спочатку запропонуємо вашій увазі очевидний простий спосіб розв'язання задачі. Він полягає у перебиранні усіх можливих прямокутних ділянок та перевірці кожної з таких ділянок на можливість розміщення ринку на ній. Такий спосіб має складність порядку $\Theta(n^6)$ (де **n** — розмір таблиці), тому він цілком прийнятний для $n \le 10$ і може дати задовільні часові показники при трошки більших **n**.

```
Program O_04_05_1;
var n,m,i,j,k,l,x1,x2,y1,y2,max:integer;
 a:array[1..20,1..20]of byte;
 f:text;
 function sum_ok(e,b,c,d:byte):boolean;
{Функція перевіряє, чи є порожньою прямокутна ділян-
ка}
  var i,j:byte;
  begin
    sum_ok:=false;
    for i:=e to c do
      for j:=b to d do
        if a[i,j]=1 then exit;
    sum_ok:=true;
  end;
begin
 assign(f, 'market.dat');
 reset(f);
 read(f,n,m);
 for i:=1 to n do
  for j:=1 to m do read(f,a[i,j]);
 close(f);
 max:=0;
 {Наступині вкладені цикли перебирають всі мож-
ливі прямокутні ділянки, починаючи з найбільшої, на
наявність у них забудов. Для оптимізації перебору
перевіряється забудова вершин прямокутника, також
при знаходженні кожної такої ділянки відбувається
припинення останнього циклу.}
 for i:=1 to n-1 do
  for j:=1 to n-1 do
   for k:=n downto 2 do
    for l:=m downto 2 do
     if a[i,j]+a[k,j]+a[i,l]+a[j,l]=0
     then if sum_ok(i,j,k,l)
          then begin
                 if max<(k-i+1)*(l-j+1)
                 then begin
                        x1:=i;y1:=j;x2:=k;y2:=l;
                        max:=(k-i+1)*(l-j+1);
```

```
                    end;
                  break;
                end;
  assign(f, 'market.sol');
  rewrite(f);
  writeln(f,max);
  writeln(f,x1,' ',y1);writeln(f,x2,' ',y2);
  close(f);
end.
```

Пропонуємо вашій увазі більш *оптимальне* розв'язання даної задачі. Введемо до розгляду матрицю K, яку формуватимемо поступово зверху донизу. Елемент K_{ij} зберігатиме висоту неперервного стовпчика з нулів, основою якого є клітинка (i,j). Маючи значення одного рядка матриці K, можна підрахувати площу вільної прямокутної ділянки, основою якої є клітинки $(i,k)...(i,l)$: ця площа дорівнюватиме $(l-k)\cdot\min\limits_{j\le s\le k}\{K_{ij}\}$. Серед усіх можливих прямокутників тоді слід обрати один із максимальною площею.

Для визначення координат лівого верхнього та правого нижнього кутів прямокутника максимальної площі слід щоразу запам'ятовувати позицію, на якій досягався поточний максимум площі.

Це розв'язання має складність порядку $\Theta(n^3)$, тому вкладається у часові межі навіть за $n \le 100$.

```
const maxn = 20;
var n, m, i, j, L, TLi, TLj, BRi, BRj: byte;
    A, K: array [0..maxn,0..maxn] of integer;
    maxs: integer;
    F:text;
begin
  assign(F, 'market.dat');
  reset(F);
  read(F,n,m);
  for i:=1 to n do
    for j:=1 to m do
     read(F,A[i,j]);
  close(F);
  max:=0;
  for i:=1 to n do
```

```
      for j:=1 to m do
        if (A[i,j]=1)
        then begin
               K[i,j]:=K[i-1,j]+1;
               h:=K[i,j]; L:=j;
               while (h>0) do
               begin
                 if (h>K[i,L])
                 then h:=K[i,L];
                 if ((j-1)*h>maxs)
                 then begin
                        maxs:=(j-1)*h;
                        TLi:=i-h; TLj:=1;
                        BRi:=i; BRj:=j;
                      end;
               end;
             end;
    assign(F, 'market.sol');
    rewrite(F);
    writeln(F,maxs);
    if (maxs<>0)
    then begin
           writeln(F,TLi,' ',TLj);
           writeln(F,BRi,' ',BRj);
         end;
    close(F);
  end.
```

Задача 2. **Найбільший добуток**

Дано **N** цілих чисел. Вибрати з них три таких числа, добуток яких максимальний.

Вхідний файл: у першому рядку число **N** — кількість чисел у послідовності ($3 \le N \le 10^6$). У другому рядку записана сама послідовність **N** цілих чисел, кожне з яких по модулю не перевищують 30 000.

Вихідний файл: у вихідному файлі є три шуканих числа в будь-якому порядку. Якщо існує кілька різних трійок чисел, що дають максимальний добуток, виведіть будь-яку з них.

Наприклад:

DOBUTOK.DAT	DOBUTOK.SOL
9 3 5 1 7 9 0 9 -3 10	9 10 9
3 -5 -30000 -12	-5 -3000 -12

Пояснення до розв'язань

Ця задача є тривіальною, вона розрахована на уважність учнів. Необхідно пам'ятати, що добуток двох від'ємних чисел є числом додатним. Отож, необхідно у прорядкованій послідовності порівняти добутки двох найменших від'ємних чисел і двох передостанніх найбільших чисел. Оскільки впорядковувати послідовність з 10^6 елементів недоцільно з урахуванням часового обмеження, обмежимося лише послідовністю з п'яти елементів, упорядкованих за спаданням, яка містить три найбільших та два найменших елементи послідовності.

Увага! При цьому будуть враховані всі випадки, зокрема відсутність від'ємних або додатних чисел у послідовності.

```
Program O_04_05_2;
var f:text;
  i,n,m:longint;
  mas:array[1..5] of longint;
  begin
   assign(f, 'dobutok.dat');
   reset(f);
   read(f,n);
   fillchar(mas,sizeof(mas),0);
   mas[1]:=-maxlongint;
   mas[2]:=-maxlongint;
   mas[3]:=-maxlongint;
   mas[4]:=maxlongint;
   mas[5]:=maxlongint;
   for i:=1 to n do
   begin
     read(f,m);
     if (m>mas[1])
     then begin
```

```
        mas[3]:=mas[2];mas[2]:=mas[1];mas[1]:=m;
          end
      else if m>mas[2]
           then begin
                  mas[3]:=mas[2]; mas[2]:=m;
                end
           else if m>mas[3]
                then mas[3]:=m;
      if (m<mas[5])
      then begin mas[4]:=mas[5];mas[5]:=m;end
      else if m<mas[4]
           then mas[4]:=m;
    end;
    close(f);
    assign(f, 'dobutok.sol');
    rewrite(f);
    if mas[2]*mas[3]>mas[4]*mas[5]
    then writeln(f,mas[1], '',mas[2], '',mas[3])
    else writeln(f,mas[1], '',mas[4], '',mas[5]);
    close(f);
  end.
```

Задача 3. **Доміно**

Як правило, комплект доміно має 28 пластинок. Якби кількість крапок на пластинках змінювалось не від 0 до 6, а від 0 до деякого K, то, вочевидь, кількість пластинок була б іншою. З'ясуйте, скільки пластинок має комплект доміно, випущений фірмою CHASprotect, написавши програму **DOMINO.PAS**.

Вхідний файл: **DOMINO.DAT** у єдиному рядку містить праву межу зміни вічок на пластинах у вигляді натурального числа K ($K \le 100000000$).

Вихідний файл: **DOMINO.SOL** містить число — кількість пластинок, яка входить у даний комплект доміно.

Наприклад:

DOMINO.DAT	DOMINO.SOL
7	36
11	78

Пояснення до розв'язань

Ця задача є суто комбінаторною.

Не знаючи стандартних формул комбінаторики, можна було б міркувати так. На одній пластинці доміно міститься два числа від 0 до K, а у наборі усі пластинки різні. Для визначеності встановимо, що в усіх доміношках ліве число не перевищує праве (адже довільну доміношку можна розвернути для виконання цієї умови). Тоді якщо ліворуч на пластинці написане число $x \in [0,K]$, то праворуч може бути довільне з чисел $[x,K]$. Перебираючи усі можливі значення x, підсумуємо кількість можливих пластинок доміно, у яких лівим числом є число x: для фіксованого x ця кількість дорівнює $(K-x+1)$. Тому загальна кількість доміношок $S_k = \sum_{x=0}^{K}(K-x+1) = \sum_{x=1}^{K+1} x = \frac{(K+1)(K+2)}{2}$. Остання формула — сума арифметичної прогресії. Підставляючи у цей вираз стандартне значення $K=6$, одержимо $S_6 = \frac{7\cdot 8}{2} = 28$, що відповідає дійсності.

З урахуванням обмеження $K \le 10^6$ можна при обчисленнях скористатися типом *extended*, який може зберігати до 20 значущих цифр.

```
Program O_04_05_3;
{$N+}
var K: extended;f:text;
begin
  assign(F, 'domino.dat');
  reset(F);
  readln(F,K);
  close(F);
  assign(F, 'domino.sol');
  rewrite(F);
  writeln(F,round((K+1)*(K+2)/2));
  close(F);
end.
```

Задача 4. **Офіцери**

За даними розвідки серед шести офіцерів A, B, C, D, E, F є три полковники, два майори і один капітан. Один з офіцерів — зв'язківець, решта танкісти та артилеристи. Єдиний рядок вхідного файла **OFICERS.DAT** містить у вказаному порядку 8 літер латиниці **a1, a2, . . . , a8**, не розділених пропусками, якими позначено офіцерів (деякі літери з переліку A, B, C, D, E, F можуть повторюватися, а деякі — не зустрічатися взагалі). Відомо (у кожному з поданих далі *простих* речень офіцери, про яких ідеться, різні), що **a1** зобов'язаний першим привітати **a2** (майор зобов'язаний першим привітати полковника, а капітан — і полковника, і майора), **a3** і **a4** — в одному званні, **a5** і **a6** — в одному роді військ. Полковник, майор і **a7** — танкісти **a8** і капітан — артилеристи. Створіть програму **OFICERS.PAS**, яка у вихідний файл **OFICERS.SOL** запише, хто є хто серед офіцерів. Потрібно через кому перерахувати всіх офіцерів у алфавітному порядку, вказавши через дефіс перші літери звання та роду військ) — по одному рядку на кожен варіант відповіді.

Наприклад, якщо файл **OFICERS.DAT** має вигляд **ABDECFDB**, то файл **OFICERS.SOL** має два рядки

A к-а, B м-а, C м-т, D п-т, E п-з, F п-т.

A к-а, B м-а, C п-т, D п-т, E п-з, F м-т.

Останній рядок читається так: A капітан-артилерист, B майор-артилерист, C полковник-танкіст, D полковник-танкіст, E полковник-зв'язківець, F майор-танкіст. Якщо розвідувальні дані хибні (призводять до логічних суперечностей), то файл **OFICERS.SOL** порожній.

Наприклад:

OFICERS.DAT	**OFICERS.SOL**
ABDECFDB	A к-а, B м-а, C м-т, D п-т, E п-з, F п-т. A к-а, B м-а, C п-т, D п-т, E п-з, F м-т.
FEDCBABA	(Порожній файл, тобто розвідувальні данні хибні)

Пояснення до розв'язань

Спробуймо зробити деякі логічні висновки з висловлювань, описаних в умові задачі:

1. Серед усіх офіцерів є один зв'язківець, два артилеристи й три танкісти.
2. Один з офіцерів — капітан-артилерист.
3. Один з офіцерів — полковник-танкіст.
4. Один з офіцерів — майор-танкіст.
5. **a1** не є полковником, а **a2** не є капітаном.
6. Зв'язківець є або майором, або полковником.

З урахуванням цих умов організуємо оптимізований перебір варіантів розподілу родів військ та військових звань між офіцерами.

```
Program O_04_05_4;
const zv: array [1..3] of char = ('к', 'м', 'п');
      rv: array [1..3] of char = ('з', 'т', 'а');
var a1,a2,a3,a4,a5,a6,a7,a8: char;
    Z, R: array ['A'..'F'] of byte;
    cap_art, pol_tan, maj_tan, pol2: char;
procedure Check;
var i: Char;
    tank: array [1..3] of boolean;
begin
 { перевіряємо, чи ті люди що треба є танкістами }
  fillchar( tank, 3, false );
  for i:='A' to 'F' do
    if ( R[i] = 2 )
    then  begin
            if ( Z[i] = 3 ) then tank[1]:=true;
            if ( Z[i] = 2 ) then tank[2]:=true;
            if ( i = a7 ) then tank[3]:=true;
          end;
  if tank[1] and tank[2] and tank[3] and
     (Z[a1]<Z[a2])and(Z[a3]=Z[a4])and
     (R[a5]=R[a6])and(R[a7]=2)and
     (R[a8]=3)
  then begin
       for i:='A' to 'E' do
      write(i,' ',zv[Z[i]], '-',rv[R[i]], ',');
     writeln('F',zv[Z['F']], '-',rv[R['F']], '.');
       end;
end;
procedure MakePerebor;
var zva, pol_maj: char;
```

```
    maj_pol: byte;
begin
  for zva:='A' to 'F' do      { зв'язківець }
    if (R[zva]=0)
    then begin
           R[zva]:=1;
           for maj_pol:=2 to 3 do
{ зв'язківець - або майор, або полковник }
           begin
             Z[zva]:=maj_pol;
             pol_maj:='A';
             while (pol_maj<'F') and
                   (Z[pol_maj]<>0)
             do pol_maj:=chr(ord(pol_maj)+1);
{ останній - чи то танкіст, чи то артилерист }
             Z[pol_maj]:=5-maj_pol;
             R[pol_maj]:=2;  { (а) танкіст }
             if (Z[pol_maj]=2) or
                ((pol_maj>pol_tan) and
                ((R[pol2]<>2)or(pol_maj>pol2)))
             then  Check;
             R[pol_maj]:=3;  { (б) артилерист }
             if (Z[pol_maj]=2) or (R[pol2]<>3)
                 or (pol_maj>pol2)
             then  Check;
             Z[pol_maj]:=0;   R[pol_maj]:=0;
             Z[zva]:=0;
           end;
           R[zva]:=0;
           end;
end;
var F:text;
begin
  assign(F, 'officers.dat');
  reset(F);
  read(F,a1,a2,a3,a4,a5,a6,a7,a8);
  close(F);
  assign(F,'officers.sol');
  rewrite(F);
  FillChar(Z,SizeOf(Z),0);
```

```
  FillChar(R,SizeOf(R),0);
  for cap_art:='A' to 'F' do { капітан-артилерист }
    if not (cap_art in [a2,a8])
    then begin
           Z[cap_art]:=1;  R[cap_art]:=3;
           for pol_tan:='A' to 'F' do
{ полковник-танкіст }
             if not (pol_tan in [a1,cap_art])
             then begin
                    Z[pol_tan]:=3;
                    R[pol_tan]:=2;
             for maj_tan:='A' to 'F' do
{ майор-танкіст }
           if not (maj_tan in [pol_tan,cap_art])
              then begin
                 Z[maj_tan]:=2;  R[maj_tan]:=2;
                    for pol2:='A' to 'F' do
                      if (Z[pol2]=0)
                      then begin
                             Z[pol2]:=3;
   if (pol_tan<pol2)and((pol_tan=a7)or(pol2=a7))
      then begin
           R[pol2]:=2;  { (а) полковник-танкіст }
            MakePerebor;
           end;
      if (pol2<>a8)
      then begin
           R[pol2]:=3; { (б) полковник-артилерист }
            MakePerebor;
           end;
         Z[pol2]:=0;  R[pol2]:=0;
        end;
     Z[maj_tan]:=0;  R[maj_tan]:=0;
     end;
     Z[pol_tan]:=0;  R[pol_tan]:=0;
    end;
    Z[cap_art]:=0;  R[cap_art]:=0;
  end;
  close(F);
end.
```

Задача 5. **Квитки до театру**

За квитками на прем'єру нового мюзиклу вишикувалась черга з **N** покупців, кожен з яких хоче придбати 1 квиток. На всю чергу працює тільки одна каса, тому черга просувається дуже повільно. Кмітливі любителі театру помітили, що коли касир продає кілька квитків в одні руки, черга просувається швидше, ніж тоді, коли ці самі білети продаються по одному. Тому вони запропонували кільком людям, які стоять один за одним, віддавати гроші першому з них, щоб придбати квитки на всіх.

З метою запобігання спекуляції касир продавав не більше 3 квитків в одні руки, тому домовитись таким чином між собою могли тільки 2 або 3 покупці, які стояли поруч.

Відомо, що на продаж **i**-й людині з черги одного квитка касир витрачає **Ai** секунд, на продаж двох квитків — **Bi** секунд, трьох квитків — **Ci** секунд. Напишіть програму, яка підрахує мінімальний час, необхідний для обслуговування всіх покупців.

Зверніть увагу, що квитки на групу людей, які об'єдналися, завжди купує перший з них, і зайвих квитків ніхто не купує.

Вхідний файл: містить число **N** — кількість покупців в черзі ($1 \le N \le 5000$). Далі іде **N** трійок натуральних чисел **Ai , Bi , Ci**. Кожне з цих чисел не перевищує 3600. Люди в черзі нумеруються починаючи від каси.

Вихідний файл: виведіть одне число — в секундах, за яке касир може продати квитки всім бажаючим.

KVITKI.DAT	KVITKI.SOL
5 5 10 15 2 10 15 5 5 5 20 20 1 20 1 1	12
2 3 4 5 1 1 1	4

Пояснення до розв'язань

Ця задача розв'язується за вже знайомим нам принципом динамічного програмування.

Розглянемо хвіст черги з останніх k людей — тобто від N-ої до $(N-k+1)$-ої людини. Позначимо F_k мінімальний час, за який касир може обробити чергу з останніх k людей. У першої людини з цієї частини черги є три альтернативи: купувати квиток лише для себе, купувати для себе і наступного у черзі або ж купувати для себе і двох наступних у черзі. Якщо він обере першу альтернативу, то мінімальний час, за який може бути оброблена черга, складатиме $A_{N-k+1}+F_{k-1}$. Другій альтернативі відповідає час $B_{N-k+1}+F_{k-2}$, а третій $C_{N-k+1}+F_{k-3}$. Серед цих альтернатив слід обрати ту, для якої час найменший.

```
Program O_04_05_5;
var i, n: integer;
    a, b, c, f: array [1..5002] of integer;
    fi:text;
begin
  assign(fi, 'kvitki.dat');
  reset(fi);
  read(fi,n);
  for i:=1 to n do
   read(fi,a[i],b[i],c[i]);
  close(fi);
  f[n+1]:=0; f[n+2]:=0; f[n+3]:=0;
  for i:=n downto 1 do
  begin
    f[i]:=a[i]+f[i+1];
    if (f[i]>b[i]+f[i+2])
    then f[i]:=b[i]+f[i+2];
    if (f[i]>c[i]+f[i+3])
    then f[i]:=c[i]+f[i+3];
  end;
 assign(fi, 'kvitki.sol');
 rewrite(fi);
 write(fi,F[1]);
 close(fi);
end.
```