novática
UPGRADE Revista de la Asociación de Técnicos de Informática

Novática, revista fundada en 1975 y decana de la prensa informática española, es el órgano oficial de expresión y formación continua de **ATI** (Asociación de Técnicos de Informática). *Novática* edita también *Upgrade*, revista digital de **CEPIS** (*Council of European Professional Informatics Societies*), en lengua inglesa, y es miembro fundador de **UPENET** (*UPGRADE European NETwork*)

<http://www.ati.es/novatica/>
<http://www.upgrade-cepis.org/>

ATI es miembro fundador de **CEPIS** (*Council of European Professional Informatics Societies*) y es representante de España en **IFIP** (*International Federation for Information Processing*); tiene un acuerdo de colaboración con **ACM** (*Association for Computing Machinery*), así como acuerdos de vinculación o colaboración con **AdaSpain**, **AI2** y **ASTIC.**

Coordinación Editorial, Redacción Central y Redacción ATI Madrid
Padilla 66, 3º, dcha., 28006 Madrid
Tlfn.914029391; fax.913093685 <novatica@ati.es>
Composición, Edición y Redacción ATI Valencia
Av. del Reino de Valencia 23, 46005 Valencia
Tlfn./fax 963330392 <secreval@ati.es>
Administración y Redacción ATI Cataluña
Via Laietana 41, 1º, 1ª, 08003 Barcelona
Tlfn.934125235; fax 934127713 <secregen@ati.es>
Redacción ATI Andalucía
Isaac Newton, s/n, Ed. Sadiel,
Isla Cartuja 41092 Sevilla, Tlfn./fax 954460779 <secreand@ati.es>
Redacción ATI Aragón
Lagasca 9, 3-B, 50006 Zaragoza.
Tlfn./fax 976235181 <secreara@ati.es>
Redacción ATI Asturias-Cantabria <gp-astucant@ati.es>
Redacción ATI Castilla-La Mancha <gp-clmancha@ati.es>
Redacción ATI Galicia
Recinto Ferial s/n, 36540 Silleda (Pontevedra)
Tlfn.986581413; fax 986580162 <secregal@ati.es>
Suscripción y Ventas
<http://www.ati.es/novatica/interes.html>,o en ATI Cataluña o ATI Madrid
Publicidad
Padilla 66, 3º, dcha., 28006 Madrid
Tlnf.914029391; fax.913093685 <novatica.publicidad@ati.es>
Imprenta
9 Impressió S.A., Juan de Austria 66, 08005 Barcelona.
Depósito legal: B.15.154-1975 -- ISSN: 0211-2124; CODEN NOVAEC
Portada: Antonio Crespo Foix / © ATI 2004
Diseño: Fernando Agresta / © ATI 2004

Nº 171, septiembre-octubre 2004, año XXX

sumario

Monografía del próximo número: "Criptografía"

Problema G: Diseño de suelos

Juan Céspedes Prieto[1], Antonio Fernández Anta[1], Ángel Herranz Nieva[2]
[1] Universidad Rey Juan Carlos; [2] Facultad de Informática, Universidad Politécnica de Madrid

<cespedes,anto@gsyc.escet.urjc.es>
<aherranz@fi.upm.es>

El enunciado de este problema apareció en el número 170 de ***Novática*** (julio-agosto 2004, p. 72). Es el problema G de los planteados en el I Concurso Universitario de la Comunidad Autónoma de Madrid (CUPCAM 2003)

1. Resumen del problema

El problema presentado en el número anterior pedía encontrar el número de todos los posibles diseños distintos que se pueden elegir para un suelo dado un determinado conjunto de baldosas. Este problema tiene una fuerte componente combinatoria. Disponemos de sólo 4 tipos de baldosas y el diseño debe ser simétrico.

2. Contando diseños

Si tuviéramos tanta cantidad de baldosas de cada tipo como quisiéramos, bastaría con contar todos los posibles diseños simétricos. Para ello podemos observar que en un diseño simétrico basta fijar una mitad del mismo para tener el diseño completo. Claramente, si la longitud es impar la fila central debe formar parte de esa 'mitad' que hay que fijar. Entonces, para el caso concreto de una habitación de dimensiones $l \times w$, el número de baldosas a fijar es $\lceil l/2 \rceil \times w$. Como cada baldosa debe ser de uno de 4 posibles tipos, el número total de diseños (D_{tot}) se calcula fácilmente así:

$$D_{tot}(l,w) = 4^{\lceil l/2 \rceil w}. \quad (1)$$

Sin embargo, tenemos limitado el número de baldosas de cada tipo (t_i). Eso impide que podamos usar la fórmula 1 directamente. Sin embargo sí se podría plantear el generar todos estos casos y no contar aquéllos que no cumplan con las restricciones en número de baldosas de algún tipo. Como en el caso peor nos dicen que la superficie es de 33 pies cuadrados, y éste se puede dar con una habitación de 33×1, en el caso peor estaríamos hablando de 4^{33} casos, que es un número enorme y hace inviable esta solución.

3. Contando con más cuidado

Por lo tanto, vamos a tener que contar con más cuidado y tener en cuenta el número de baldosas disponible de cada tipo. Primero debemos observar que cuando consideramos solamente la mitad del diseño, como hacemos arriba, fijar una baldosa a un tipo en realidad implica consumir dos baldosas de ese tipo, excepto si la longitud es impar y estamos fijando la fila central. Por ello vamos a separar la superficie de la habitación en dos partes, la fila central y el resto.

Entonces, si tenemos que diseñar una superficie de área s, y hemos fijado un reparto de sus baldosas entre los cuatro tipos, donde b_i ($0 \le b_i \le t_i$) es el número de baldosas elegido de tipo i tal que

$$\sum_{i=1}^{4} b_i = s \ ,$$

podemos usar la fórmula de las permutaciones con repetición para calcular el número de posibles diseños (D_{lim}) con ese reparto:

$$D_{lim}(s,b,b_1,b_2,b_3,b_4) = \frac{s!}{\prod_{i=1}^{4} b_i!} \quad (2)$$

Entonces, sumando estos valores para todos los posibles repartos tendríamos el número total de diseños para esa superficie. Nótese que para un área de s el número máximo de posibles repartos se puede calcular como

$$\sum_{i=0}^{s} \sum_{j=i}^{s} s\text{-}j+1 \le (s+1)^3, \quad (3)$$

que para s pequeño es una cantidad razonable.

4. Programando

Este análisis da lugar al programa que presentamos. En el mismo se distingue si la longitud de la habitación es un número par o impar. En el primer caso simplemente busca todos los posibles repartos para la mitad de la superficie, teniendo en cuenta, como hemos dicho, que cada posición en realidad consume dos de las baldosas disponibles, y se suma el número de diseños con cada reparto. Sin embargo, si la longitud es impar, se multiplica el número de diseños para la fila central por el número de diseños total para el resto de la habitación. De nuevo, sólo se considera la mitad del resto.

```
#include <stdio.h>
#define MIN(a,b) ((a<b)?(a):(b))
#define MAX(a,b) ((a>b)?(a):(b))

long int fact(int x)
{ int i;
  long int r=1;

  for (i=2; i<=x; i++) r *= i;
  return r;
}

int even_length(size,t1,t2,t3,t4)
  int size,t1,t2,t3,t4;
{ int b1,b2,b3;
  long int count=0, fsize;

  fsize=fact(size);
  for (b1=0;
       b1<=MIN(size,t1);
       b1++)
    for (b2=0;
         b2<=MIN(size-b1,t2);
         b2++)
      for (b3=0;
           b3<=MIN(size-b1-b2,t3);
           b3++)
        if (t4 >= size-b1-b2-b3)
          count +=
            fsize
            /(fact(b1)
              *fact(b2)
              *fact(b3)
              *fact(size-b1-b2-b3));
  return count;
}

int odd_length(length,width,
               t1,t2,t3,t4)
  int length,width,t1,t2,t3,t4;
{ int b1,b2,b3,size;
  long int count=0, fwidth;

  fwidth=fact(width);
  size=(length-1)*width/2;
  for (b1=0;
       b1<=MIN(width,t1);
       b1++)
    for (b2=0;
         b2<=MIN(width-b1,t2);
         b2++)
      for (b3=0;
           b3<=MIN(width-b1-b2,t3);
           b3++)
        if (t4 >= width-b1-b2-b3)
          count +=
            (fwidth
             /(fact(b1)
```

Nota del Editor: con la solución de este problema de programación concluimos la serie de los planteados en el I Concurso Universitario de la Comunidad Autónoma de Madrid (CUPCAM 2003), serie Iniciada en el número 166 (noviembre-diciembre 2003) y que hemos podido ofrecer a nuestros lectores gracias al generoso y continuado esfuerzo de un grupo de docentes informáticos encabezado por nuestro colaborador ***Cristóbal Pareja Flores***. A todos ellos expresamos nuestro más sincero agradecimiento.

```
                *fact(b2)
                *fact(b3)
                *fact(width-b1-b2-b3)))
            * even_length(
                size,
                (t1-b1)/2,
                (t2-b2)/2,
                (t3-b3)/2,
                (t4-(width-b1-b2-b3))
                  / 2);
  return count;
}

main() {
  long int l, w, t1, t2, t3, t4;
  long int d;

  scanf("%d", &l);
  while (l != 0) {
    scanf("%d %d %d %d %d",
          &w, &t1, &t2, &t3, &t4);
    if ((l % 2) == 0) {
      d = even_length(l*w/2,
                      t1/2,
                      t2/2,
                      t3/2,
                      t4/2);
    }
    else {
      d = odd_length(l,
                     w,
                     t1,
                     t2,
                     t3,
                     t4);
    }
    printf("Número de diseños: %d\n",
           d);
    scanf("%d", &l);
  }
}
```

5. Desbordamiento

Tras ejecutar el programa anterior con distintos casos de prueba nos encontramos con la siguiente situación:

```
$ ./tiles
7 4 20 20 20 20
Número de diseños: -762568
0
$
```

Esto indica que nuestro programa sufre desbordamientos. Puesto que la superficie de la habitación es como máximo de 33 pies cuadrados, existe la posibilidad de que nuestra solución acabe intentando calcular el factorial de 33 (8683317618811886495 51819440128e7[1]) algo que garantizaría resultados incorrectos incluso utilizando mayor precisión (ver la definición de la macro `ULLONG_MAX` en `limits.h`).

A continuación mostramos un programa que supera este problema precomputando los números combinatorios necesarios (*triángulo de Pascal*) y en el que se ha introducido una variación en el cálculo del número de diseños con un algoritmo recursivo sobre el número de huecos sin cubrir y el tipo de baldosa (evitando un código dependiente de los tipos de baldosas).

```
#include <stdio.h>
#define MAX_AREA 33
#define MAX_TIPOS 4

/*
 * Indice 0 indica huecos por cubrir.
 * Indices mayores que 0 indican
 * baldosas disponibles de un tipo.
 */
typedef unsigned int
  disponibles_t[MAX_TIPOS+1];
/* Números combinatorios */
unsigned C[MAX_AREA+1][MAX_AREA+1];

/* Triángulo de pascal */
void pascal()
{
  unsigned n, m;
  for (n = 0; n <= MAX_AREA; n++) {
    C[n][0] = 1;
    for (m = 1; m < n; m++) {
      C[n][m] =
        C[n-1][m-1] + C[n-1][m];
    }
    C[n][m] = 1;
  }
}

/* Diseños */
long unsigned D(unsigned tipo,
                disponibles_t disp)
{
  unsigned huecos = disp[0];
  long unsigned disegnos;

  if (huecos == 0) {
    /* Suelo cubierto */
    disegnos = 1;
  }
  else if (tipo == 0) {
    /* Suelo imposible de cubrir */
    disegnos = 0;
  }
  else {
    int usadas;

    disegnos = 0;
    for (usadas = 0;
         usadas <= disp[tipo]
           && usadas <= huecos;
         usadas++) {
      disp[0] = huecos - usadas;
      disegnos +=
        D(tipo - 1, disp)
        * C[huecos][usadas];
    }
  }
  return disegnos;
}

int main() {
  unsigned l, w, d1, d2, d3, d4;
  disponibles_t disp;

  pascal();

  scanf("%d", &l);
  while (l != 0) {
    scanf("%d %d %d %d %d",
          &w, &d1, &d2, &d3, &d4);
    disp[0] = l * w / 2;
    disp[1] = d1 / 2;
    disp[2] = d2 / 2;
    disp[3] = d3 / 2;
    disp[4] = d4 / 2;
    if ((l % 2) == 0) {
      printf("\
Número de diseños: %lu\n",
             D(MAX_TIPOS,disp));
    }
    else {
      fprintf(stderr,
              "\
No funciona para longitud impar:(\n");
      return 1;
    }
    scanf("%d", &l);
  }
  return 0;
}
```

Como se puede ver, se han dejado al lector un par de ejercicios interesantes:

- Añadir al programa la capacidad de calcular los números de diseños para longitudes impares sin introducir un código dependiente de los tipos de baldosas.
- Evitar tener que recalcular números combinatorios utilizando la propiedad $C^{n,m}=C^{m,n}$.

Agradecimientos

Queremos terminar con unas palabras de agradecimiento a Cristobal Pareja Flores por todas sus aportaciones a estos comentarios.

Nota

[1] Calculado en el intérprete de Haskell Hugs mediante la expresión `let f n = if n==1 then 1 else n*f(n-1) in f 33`