

Novática, revista fundada en 1975, es el órgano oficial de expresión y formación continua de ATI (Asociación de Técnicos de Informática)

ATI es miembro de CEPIS (Council of European Professional Informatics Societies) y tiene un acuerdo de colaboración con ACM (Association for Computing Machinery). Tiene asimismo acuerdos de vinculación o colaboración con AdaSpain, AI² y ASTIC

<http://www.ati.es/novatica/>

CONSEJO ASESOR DE MEDIOS DE COMUNICACION

Pere Lluís Barbarà, Rafael Fernández Calvo, José Gómez, Manuel Orti Mezquita, Nacho Navarro, Fernando Sanjuán de la Rocha (Presidente), Miquel Sarries, Carlos Sobrino Sánchez, Manuel Solans

Coordinación Editorial
Rafael Fernández Calvo <rfcalvo@ati.es>

Composición y autoedición
Jorge Llácer

Administración
Tomás Brunete, Joan Aguiar, María José Fernández

SECCIONES TÉCNICAS: COORDINADORES

Arquitecturas
Antonio Gonzalez Colás (DAC-UPC) <antonio@ac.upc.es>

Bases de Datos
Mario G. Plattini Velthuis (EUI-UCLM) <mpiatini@inf-cr.uclm.es>

Calidad del Software
Juan Carlos Granja (Universidad de Granada) <jcgranja@goliat.ugr.es>

Derecho y Tecnologías
Isabel Hermando Collazos (Fac. Derecho de Donostia, UPV) <ihermando@legaltek.net>

Enseñanza Universitaria de la Informática
Cristóbal Pareja Flores (Dep. Sistemas Informáticos y Programación-UCM) <cpareja@sip.ucm.es>

Euro/Efecto 2000
Joaquín Ríos Boutin <jrios@ati.es>

Informática Gráfica
Roberto Vivó (Eurographics, sección española) <rvivo@dsic.upv.es>

Informática Médica
Valentín Masero Vargas (DI-UNEX) <vmasero@umex.es>

Ingeniería del Software
Luis Fernández (PRIS-E.I./UEM) <lufern@dpris.esi.uem.es>

Inteligencia Artificial
Federico Barber, Vicente Botti (DSIC-UPV) <fjbotti, fbarber@dsic.upv.es>

Interacción Persona-Computador
Julio Abascal González (FI-UPV) <julio@si.ehu.es>

Internet
Alonso Álvarez García (TID) <alonso@ati.es>

Llengua e Informàtica
Llorenç Pagès Casas (Atlante) <pages@ati.es>

Lenguajes informáticos
Andrés Marín López (Univ. Carlos III) <amarin@it.uc3m.es>

Libertades e Informática
J. Ángel Velázquez (ESCET-URJC) <a.velazquez@escet.urjc.es>

Lingüística computacional
Alfonso Escolano (FIR-Univ. de La Laguna) <aescolan@ull.es>

Lingüística computacional
Xavier Gómez Guinovart (Univ. de Vigo) <xgomez@uvigo.es>

Manuel Palomar (Univ. de Alicante) <mpalomar@dlsi.ua.es>

Profesión informática
Rafael Fernández Calvo (ATI) <rfcalvo@ati.es>

Seguridad
Miquel Sarries Grinyó (Ayto. de Barcelona) <msarries@ati.es>

Sistemas de Tiempo Real
Javier Areitio (Redes y Sistemas, Bilbao) <jareitio@orion.deusto.es>

Sistemas de Tiempo Real
Alejandro Alonso, Juan Antonio de la Puente (DIT-UPM) <jaalonso.jp puente@diti.upm.es>

Software Libre
Jesús M. González Barahona, Pedro de las Heras Quiros (GSYC, URJC) <jgb, pheras@gsyc.escet.urjc.es>

Tecnología de Objetos
Esperanza Marcos (URJC) <e.marcos@escet.urjc.es>

Tecnologías para la Educación
Gustavo Rossi (LIFIA-UNLP, Argentina) <gustavo@sol.info.unpl.edu.ar>

Tecnologías para la Educación
Benita Compostela (F. CC. PP.- UCM) <benita@principes.es>

Tecnologías y Empresa
Josep Sales Rufí (ESPIRAL) <jsales@pie.xtec.es>

TIC para la Sanidad
Pablo Hernández Medrano (Meta4) <pabloh@meta4.es>

TIC para la Sanidad
Valentín Masero Vargas (DI-UNEX) <vmasero@umex.es>

Las opiniones expresadas por los autores son responsabilidad exclusiva de los mismos. Novática permite la reproducción de todos los artículos, salvo los marcados con © o copyright, debiéndose en todo caso citar su procedencia y enviar a Novática un ejemplar de la publicación.

Coordinación Editorial y Redacción Central (ATI Madrid)
Padilla 66, 3º, dcha., 28006 Madrid
Tf:914029391; fax:913093685 <novatica@ati.es>

Composición, Edición y Redacción ATI Valencia
Palomino 14, 2º, 46003 Valencia
Tf./fax:963918531 <secreval@ati.es>

Administración, Suscripciones y Redacción ATI Cataluña
Via Laietana 41, 1º, 08003 Barcelona
Tf:934125235; fax:934127713 <secregen@ati.es>

Redacción ATI Andalucía
Isaac Newton, s/n, Ed. Sadiel, Isla Cartuja 41092 Sevilla
Tf./fax:954460779 <secreand@ati.es>

Redacción ATI Aragón
Lagasca 9, 3-B, 50006 Zaragoza
Tf./fax:976235181 <secreara@ati.es>

Redacción ATI Asturias-Cantabria
Tf./fax:98581413; fax:986580162 <secregal@ati.es>

Redacción ATI Castilla-La Mancha
Recinto Ferial s/n, 36540 Silleda (Pontevedra)
Tf:986581413; fax:986580162 <secregal@ati.es>

Redacción ATI Galicia
Recinto Ferial s/n, 36540 Silleda (Pontevedra)
Tf:986581413; fax:986580162 <secregal@ati.es>

Publicidad: Padilla 66, 3º, dcha., 28006 Madrid
Tf:914029391; fax:913093685 <novatica.publicidad@ati.es>

Imprenta: Gráficas Sierra S.L., Atenas, 3, int. bajos, 08006 Barcelona.
Depósito Legal: B 15.154-1975
ISBN: 0211-2124; CODEN NOV AEC

Portada: Antonio Crespo Foix / © ATI 2001

SUMARIO

Editorial: Una nueva Junta Directiva de ATI para un nuevo periodo 2
 En resumen: Omnipresencia 2

Monografía: «Computación Ubicua»
 (En colaboración con **Informatik/Informatique** y **Upgrade**)
 Coordinada por **Friedemann Mattern, Manuel Ortega Cantero** y **Jesús Lorés Vidal**
Presentación: Computación Ubicua, la tendencia hacia la informatización y conexión en red de todas las cosas 3
Friedemann Mattern, Manuel Ortega Cantero, Jesús Lorés Vidal
Visión y fundamentos técnicos de la Computación Ubicua 4
Friedemann Mattern
El ensueño 8
Jakub Wejchert
Computación Ubicua: el punto de encuentro entre computación y dispositivos 12
Hans-Werner Gellersen
AULA: un sistema ubicuo de enseñanza de idiomas 16
Manuel Ortega, Maximiliano Paredes, Miguel Angel Redondo, Pedro Pablo Sánchez-Villalón, Crescencio Bravo, José Bravo
Un modelo interactivo ubicuo aplicado al patrimonio natural y cultural del área del Montsec 22
Montserrat Sendín, Jesús Lorés, Carles Aguiló, Alexandra Balaguer
Portales Internet para dispositivos de Computación Móvil y Pervasiva 26
Klaus Rindtorff, Martin Welsch
Un modelo de mundo para Sistemas Reconocedores de Posición 31
Matthias Grossmann, Alexander Leonhardi, Bernhard Mitschang, Kurt Rothermel
Un estudio comparativo de infraestructuras de comunicación para la implementación de Computación Ubicua 35
Pablo Haya, Xavier Alamán, Germán Montoro

Secciones técnicas

Calidad del Software
Una aportación a la medición de la Calidad del Software en entornos gráficos 40
Ángel Oller Segura, Juan Carlos Granja Alvarez

Enseñanza Universitaria de la Informática
Consideraciones sobre la Enseñanza de la Programación en Estudios Universitarios de Estadística e Investigación Operativa 44
Carlos Gregorio Rodríguez, Cristóbal Pareja Flores, Teresa Pérez Pérez

Ingeniería del Software
EVA: herramienta para la autoevaluación de la capacidad de una organización software 48
Javier García, Antonio de Amescua, Eva Cabrero, José Antonio Calvo-Manzano, Tomás San Felú

Libertades e Informática
Echelon hoy 52
Juan Vicente Oltra Gutiérrez

Seguridad
De mí misma libre me Dios, que del Sircam ya me libro yo (I) 56
Mª del Carmen Ugarte García

TIC para la Sanidad
SEDDIC, una aplicación al diagnóstico médico de las herramientas de libre distribución para el desarrollo de Sistemas Expertos 60
Iñigo Monedero Goicoechea, José Manuel Elena Ortega, Carlos León de Mora

Referencias autorizadas 67

Sociedad de la Información
Programar es crear
Fila y asociados 70
Concurso de Programación ACM 2000: programa G
«La venganza de Abbott»: solución 71
Pablo Sánchez Torralba

Asuntos Interiores
 Programación de Novática 75
 Cartas a Novática 76
 Normas de publicación para autores / Socios Institucionales 77

Monografía del próximo número: «Software Libre/Fuente Abierta: hacia la madurez»

Programar es crear

Pablo Sánchez Torralba

<psanchez@isys.dia.fi.upm.es>

**«La venganza de Abbott»:
solución**

El enunciado de este problema apareció en el número 151 de Novática (mayo-junio 2001, pp. 71-72). Es el programa A de los planteados en el 24º Concurso Internacional de Programación de la ACM (2000)

```

/* algoritmo: Búsqueda en anchura (BFS)
/*
/* notas: El grafo considerado no es el formado por los vértices de las intersecciones únicamente, sino aquel en el que se considera también la dirección de entrada en una intersección. Esto es así puesto que la visita de una intersección no es independiente del sentido de entrada, que condiciona las posibles salidas, y por la adyacencia del vértice en el grafo.
*/

#include <stdio.h>

#define NORTH 0      /* Equivalencias de direcciones */
#define EAST 1
#define SOUTH 2
#define WEST 3

int dirs[ 4][ 2] = {{ 0,-1} , { 1,0} , { 0,1} , {-1,0}}; /* Cambios en los índices */
/* según la dirección seguida*/

struct BFSNode {
    int x;          /* Elemento de la cola de recorrido */
    int y;          /* - Coordenada X */
    char dir;       /* - Coordenada Y */
    int length;     /* - Dirección de entrada */
    int prev;       /* - Longitud del camino (no es */
                  /* necesario para el problema) */
    int prev;       /* - Elemento anterior del camino */
};

char graph[ 9][ 9][ 4][ 5]; /* Grafo del laberinto: */
/* 9x9 numero de intersecciones */
/* x4 posibles formas de entrar */
/* x5 guardar las 4 posibles */
/* adyacencias como cadena (con /0) */

int discovered[ 9][ 9][ 4]; /* Vértices ya descubiertos */
char name[ 21]; /* Nombre del laberinto */
int startx, starty; /* Posición de comienzo */
char startdir; /* Dirección de comienzo */
int endx, endy; /* Posición final */

/*****
/* dir: convierte una dirección (como carácter) en el correspondiente índice */

int dir (char c)
{
    switch (c) {
        case 'N' :
            return NORTH;
            break;
        case 'E' :
            return EAST;
            break;
        case 'S' :
            return SOUTH;
            break;
        case 'W' :
            return WEST;
    }
}

```

```

        break;
    default:
        abort();
    }

/*****
/* todir: convierte a carácter una dirección */

char todir (int dir)
{
    switch (dir) {
        case 0:
            return 'N';
            break;
        case 1:
            return 'E';
            break;
        case 2:
            return 'S';
            break;
        case 3:
            return 'W';
            break;
        default:
            abort();
    }
}

/*****
/* valid: comprueba que un nodo es válido (sus índices están en el grafo
/* y no ha sido descubierto previamente) */

int valid (struct BFSNode node)
{
    return ((node.x >= 0) && (node.x < 9) &&
            (node.y >= 0) && (node.y < 9) &&
            !discovered[ node.y][ node.x][ dir(node.dir)]);
}

/*****
/* turn: calcula cuál es la dirección, dada la actual y el giro que se
/* realiza. Como los índices de las direcciones son consecutivos, es
/* una operación modular. */

char turn (char current, char t)
{
    int cur = dir(current);

    switch (t) {
        case 'F':
            return todir(cur);
            break;
        case 'L':
            return todir((cur + 3) % 4);
            break;
        case 'R':
            return todir((cur + 1) % 4);
            break;
        default:
            abort();
    }
}

/*****
/* BFS: Realiza el recorrido en anchura. Devuelve cierto si encontró un
/* camino, y falso si no existía. */

struct BFSNode queue[ 325];          /* Cola de vértices. Capacidad para */
                                     /* el numero de vértices distintos: */

```

```

                                                    /* 9x9x4 = 324 */
int head = 0; /* Cabecera de la cola */
int tail = 0; /* Fin de la cola */
                                                    /* Cola vacía si: head == tail */

int BFS (void)
{
    struct BFSNode current; /* Vértice actual de la búsqueda */
    struct BFSNode next; /* Siguiente vértice candidato */
    int i;
    char newdir;

    /* Primer elemento (al que se llega siguiendo la dirección inicial desde el */
    /* vértice inicial) */
    next.x = startx + dirs[ dir(startdir) ][ 0 ];
    next.y = starty + dirs[ dir(startdir) ][ 1 ];
    next.dir = startdir;
    next.length = 1;
    next.prev = -1;
    if (valid(next)) { /* Si el nuevo vértice es válido */
        queue[ tail++ ] = next; /* se inserta para procesarlo después */
        discovered[ next.y ][ next.x ][ dir(next.dir) ] = 1; /* y se marca descubierto */
    }

    while ((head < tail) && /* Mientras queden elementos */
           ((queue[ head ].x != endx) || /* y el actual no sea el destino */
            (queue[ head ].y != endy))) { /* seguimos recorriendo */
        current = queue[ head++ ]; /* Nodo actual (se extrae de la cola) */

        for (i = 0;
             i < strlen(graph[ current.y ][ current.x ][ dir(current.dir) ]);
             i++) {
            /* Construcción del siguiente vértice a visitar */
            newdir = turn(current.dir,
                          graph[ current.y ][ current.x ][ dir(current.dir) ][ i ]);
            next.x = current.x + dirs[ dir(newdir) ][ 0 ];
            next.y = current.y + dirs[ dir(newdir) ][ 1 ];
            next.dir = newdir;
            next.length = current.length + 1;
            next.prev = head - 1; /* Índice dentro de la cola */

            if (valid(next)) { /* Se inserta si es válido */
                queue[ tail++ ] = next;
                discovered[ next.y ][ next.x ][ dir(next.dir) ] = 1;
            }
        }
    }

    return head < tail; /* Devolvemos si se encontró */
}

/*****
/* print_path: Imprime el camino recursivamente. El parametro que recibe es */
/* índice sobre la cola de BFS. */
/* Devuelve el numero de vértices impresos, para controlar si hay */
/* que imprimir una nueva línea. */
int print_path (int index)
{
    int nl; /* Hay que poner nueva línea */

    if (queue[ index ].prev != -1) { /* No es el primer elemento */
        nl = print_path(queue[ index ].prev); /* se imprimen los anteriores */

        if (nl == 10) {
            printf("\n ");
            nl = 0;
        }

        printf(« (%d,%d)», queue[ index ].y + 1, queue[ index ].x + 1);
    }
}

```

```

    return nl + 1;
} else {
    printf(" (%d,%d)", starty + 1, startx + 1);
    printf(" (%d,%d)", queue[ index] .y + 1, queue[ index] .x + 1);

    return 2;
}
}

void main (void)
{
    char str[ 10] ;
    int i, j, k;
    int x, y;

    fscanf(stdin, «%s\n», name);
    while(strcmp("END", name) != 0) { /* Mientras no sea END */
        /* Inicialización del grafo */
        for (i = 0; i < 9; i ++ )
            for (j = 0; j < 9; j ++ )
                for (k = 0; k < 4; k ++ ) {
                    graph[ i][ j][ k][ 0] = '\0'; /* Adyacencia vacía */
                    discovered[ i][ j][ k] = 0; /* Y vértice no descubierto */
                }

        /* Lectura de toda la entrada */
        fscanf(stdin, «%d %d %s %d %d\n», &starty, &startx, str, &endy, &endx);
        startdir = str[ 0] ;
        startx --; /* Indexamos desde 0, no desde 1 */
        starty --;
        endx --;
        endy --;

        fscanf(stdin, "%d", &y); /* Lectura de las adyacencias */
        while (y != 0) {
            fscanf(stdin, "%d %s", &x, str); /* Vértice (x,y) */
            while (strcmp("**", str) != 0) { /* Procesar adyacencia */
                strcpy(graph[ y-1][ x-1][ dir(str[ 0])], &str[ 1] );

                fscanf(stdin, "%s", str); /* Siguiete adyacencia para este vértice */
            }

            fscanf(stdin, «%d», &y); /* Siguiete vertice */
        }

        printf("%s\n", name);

        if (BFS()) { /* Impresión del resultado */
            print_path(head);
            printf("\n");
        } else {
            printf(" No Solution Possible\n");
        }

        fscanf(stdin, "%s\n", name);
    }
}

```