



**CENTRO DE CIENCIAS BÁSICAS**  
**DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN**  
**OPTIMIZACIÓN INTELIGENTE**  
**5° "A"**

**PRÁCTICA 7: COLONIA DE HORMIGAS**

**Profesor: Aurora Torres Soto**

**Alumno: Joel Alejandro Espinoza Sánchez**

**Fecha de Entrega:** Aguascalientes, Ags., 15 de noviembre de 2020

## Práctica 7: Colonia de Hormigas

### Objetivo:

Con el desarrollo de la práctica se busca implementar un programa en el que una hormiga construya un camino para desplazarse desde su nido hasta la fuente de alimento siguiendo la metaheurística de colonia de hormigas.

### Introducción:

La Optimización por medio de Colonia de Hormigas (ACO), es una Metaheurística destinada originalmente a dar solución a problemas de optimización combinatoria.

Esta técnica se basa en el comportamiento estructurado de las colonias de hormigas, donde individuos muy simples se comunican entre sí por medio de una sustancia química denominada feromona, estableciendo el camino óptimo (mínimo) entre el hormiguero y su fuente de alimento.

En términos generales el algoritmo de colonia de hormigas consiste en la creación de distintas generaciones de hormigas que recorrerán el grafo del problema a resolver, construyendo en cada caso una solución por hormiga. Los movimientos que sigue una hormiga en un punto de tiempo específico dependen de información local (conocida como visibilidad) y de información global (feromona depositada por cada hormiga que ha transitado por una arista del grafo).

La siguiente figura muestra el algoritmo general de ACO (Ant Colony Optimization):

---

#### Algoritmo 1 Metaheurística ACO

---

```
Establecer parámetros, inicializar rastros de feromona
while (no se cumpla condición de terminación) do
    ConstruirSolucionesporHormigas
    AplicarBúsquedaLocal { Opcional }
    ActualizarFeromona
end while
```

---

De acuerdo con este algoritmo, la metaheurística consiste en una fase de inicialización de parámetros, seguida de una iteración sobre tres componentes: la construcción de soluciones por cada una de las hormigas, la mejora de la solución mediante algún mecanismo de búsqueda local (opcional) y la actualización de la feromona.

La construcción de una solución por cada hormiga se realiza a partir de un conjunto finito de componentes de una solución disponible. Por ejemplo, al recorrer los nodos de un grafo, cada hormiga inicia en un vértice (origen) y en cada paso se va agregando un nuevo nodo a la solución hasta llegar a la meta (destino).

La  $k$  – ésima hormiga se moverá del vértice  $i$  al  $j$  con una probabilidad dada por:

$$p_{ij}^k = \frac{(\tau_{ij})^\alpha (\eta_{ij})^\beta}{\sum (\tau_{ij})^\alpha (\eta_{ij})^\beta}$$

Donde  $\tau_{ij}$  es la cantidad de feromonas depositadas en la transición del estado  $i$  al  $j$ ,  $\alpha$  es un parámetro para controlar la influencia de  $\tau_{ij}$ ;  $\eta_{ij}$  es la conveniencia del estado de transición  $i - j$  (conocimiento a priori, típicamente  $\frac{1}{d_{ij}}$ ), donde  $d$  es la distancia o costo de la transición y  $\beta$  es un parámetro para controlar la influencia de  $\eta_{ij}$ .

Cuando las  $k$  hormigas han completado una solución, se debe actualizar los rastros de feromona por:

$$\tau_{ij} = (1 - \rho)\tau_{ij} + \sum_k \Delta\tau_{ij}^k$$

Donde:

$\tau_{ij}$  es la feromona depositada en la arista  $i - j$ ,  $\rho$  es el coeficiente de evaporación de la feromona y  $\Delta\tau_{ij}^k$  es la cantidad de feromona depositada por la  $k$  – ésima hormiga, típicamente dada por:

$$\Delta\tau_{ij}^k = \begin{cases} \frac{Q}{L_k} & \text{si } k \text{ usó la arista } i - j \\ 0 & \text{si } k \text{ no usó la arista } i - j \end{cases}$$

Donde  $L_k$  es el costo de la solución de la  $k$  – ésima hormiga y  $Q$  es una constante.

### **Pregunta de Investigación:**

¿Qué procedimientos de modelación y programación se necesitan para la codificación de un programa en lenguaje C que trate la colonia de hormigas en un grafo dado?

### **Predicción:**

Creo que se requerirá una elevada modelación de los datos pues creo que usaremos muchas estructuras, igualmente creo que el algoritmo funcionará muy satisfactoriamente.

### **Materiales:**

Una computadora con compilador de C.  
Una calculadora.

Dos hojas de papel.  
Lápiz o plumas.

### Método (Variables):

Dependiente: El programa con la información en y las configuraciones dadas.

Independiente: El resultado de la colonia de hormigas.

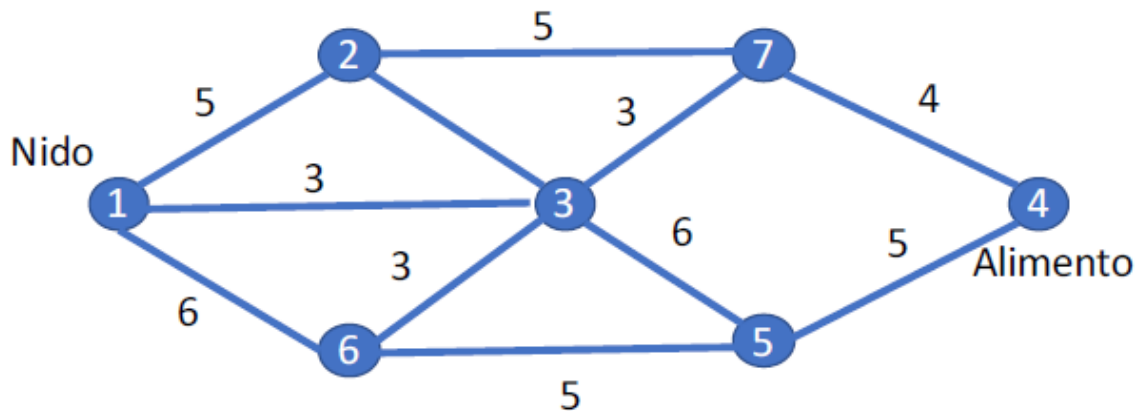
Controlada: El procedimiento de la colonia de hormigas.

### Seguridad:

Realmente no se trabajó en campo, por lo que no se corren riesgos al elaborar el experimento.

### Procedimiento:

1.- Se desarrolló un programa que permite a una hormiga desplazarse desde su nido (1) hasta la fuente de alimento (4) para el grafo siguiente:



2.- Se calculó el valor actualizado de la feromona para las 11 aristas ejecutando una iteración con una hormiga.

4.- Se mostró el recorrido de la hormiga, el valor de la distancia de ese trayecto y el valor de la feromona de todas las aristas después de la ejecución de la iteración.

### Obtención y Procesamiento de Datos:

Se usaron los parámetros siguientes en el programa (véase anexo 1), junto con las instrucciones del procedimiento

Parámetro	Valor
$\tau_{inicial}$	0.1
Q	1
$\alpha$	1
$\beta$	1
$\rho$	0.01

Al calibrar los valores siguientes en el programa, obtenemos la siguiente vista (véase anexo 2):

```

-----
Calibración del algoritmo:
Seleccione algún número si desea cambiar su valor (o 0 para continuar):
0: Listo. Continuar
1: tau1 (Tau inicial): 0.1000
2: alpha (a): 1.0000
3: beta (a): 1.0000
4: rho (a): 0.0100
5: Q (a): 1.0000
6: qh (Cantidad de hormigas): 1
7: qv (Cantidad de vértices que posee el grafo): 7
8: autom (Modo de acción): 1

```

Igualmente, al agregar el grafo en el programa, obtenemos la siguiente estructura:

```

-----
Calibración del algoritmo:
Seleccione algún número si desea cambiar su valor (o 0 para continuar):
0: Listo. Continuar
1: graph (El grafo a evaluar):
    [0] [5] [3] [0] [0] [6] [0]
    [5] [0] [4] [0] [0] [0] [5]
    [3] [4] [0] [0] [6] [3] [3]
    [0] [0] [0] [0] [5] [0] [4]
    [0] [0] [6] [5] [0] [5] [0]
    [6] [0] [3] [0] [5] [0] [0]
    [0] [5] [3] [4] [0] [0] [0]
2: BeginEnd (Los puntos de inicio y final de cada hormiga):
    [1] [4]

```

En el programa, cuando se le solicita el llenado del grafo, un atajo muy útil puede ser usando la siguiente línea después de seleccionar la opción de modificar la estructura graph:

```

0 5 3 0 0 6 0 5 0 4 0 0 0 5 3 4 0 0 6 3 3 0 0 0 0 5 0 4 0 0 6 5 0
5 0 6 0 3 0 5 0 0 0 5 3 4 0 0 0

```

El programa se le puede accionar de dos maneras: La primera es activar un modo manual (este modo estuvo más pensado como opción de desarrollador, pero también es muy útil observar cómo el programa avanza con el procedimiento, por lo que se dejó en la versión final) que es dando el valor 0 a autom. Otra es dar otro número entero positivo a autom que significará la cantidad de iteraciones que haga el programa de manera automática arrojando resultados más condensados.

Podemos ver una captura de los resultados explícitos con el modo de acción 1:

```

En la iteración 1, la hormiga 1 produjo el camino:
[1] [2] [7] [3] [6] [5] [4]
Con un recorrido de valor 26
===== Reporte Final =====
Hemos analizado el grafo:
[0] [5] [3] [0] [0] [6] [0]
[5] [0] [4] [0] [0] [0] [5]
[3] [4] [0] [0] [6] [3] [3]
[0] [0] [0] [0] [5] [0] [4]
[0] [0] [6] [5] [0] [5] [0]
[6] [0] [3] [0] [5] [0] [0]
[0] [5] [3] [4] [0] [0] [0]

La matriz de feromonas resultante es:
[0.0990] [0.1759] [0.0990] [0.0990] [0.0990] [0.0990] [0.0990]
[0.0990] [0.0990] [0.0990] [0.0990] [0.0990] [0.0990] [0.1759]
[0.0990] [0.0990] [0.0990] [0.0990] [0.0990] [0.1759] [0.0990]
[0.0990] [0.0990] [0.0990] [0.0990] [0.0990] [0.0990] [0.0990]
[0.0990] [0.0990] [0.0990] [0.1759] [0.0990] [0.0990] [0.0990]
[0.0990] [0.0990] [0.0990] [0.0990] [0.1759] [0.0990] [0.0990]
[0.0990] [0.0990] [0.1759] [0.0990] [0.0990] [0.0990] [0.0990]

La matriz de visibilidad resultante es:
[0.0000] [0.0200] [0.0333] [0.0000] [0.0000] [0.0167] [0.0000]
[0.0200] [0.0000] [0.0250] [0.0000] [0.0000] [0.0000] [0.0200]
[0.0333] [0.0250] [0.0000] [0.0000] [0.0167] [0.0333] [0.0333]
[0.0000] [0.0000] [0.0000] [0.0000] [0.0200] [0.0000] [0.0250]
[0.0000] [0.0000] [0.0167] [0.0200] [0.0000] [0.0200] [0.0000]
[0.0167] [0.0000] [0.0333] [0.0000] [0.0200] [0.0000] [0.0000]
[0.0000] [0.0200] [0.0333] [0.0250] [0.0000] [0.0000] [0.0000]

```

Sin embargo, podemos ver un desglose de datos mayor en el modo de acción 0:

```

Hemos escogido el nodo 1 (r = 0.0098)
Revisando el nodo 1 con tabú, obtuvimos que éste no es aceptado
Tendremos que repetir la selección aleatoria por posible encrucijada (La hormiga lleva 3 repeticiones)
Tenemos las probabilidades individuales y acumuladas de que de 6 se vaya a
1: [0.0167] [0.0167]
2: [0.0000] [0.0167]
3: [0.0333] [0.0500]
4: [0.0000] [0.0500]
5: [0.0200] [0.0700]
6: [0.0000] [0.0700]
7: [0.0000] [0.0700]

Hemos escogido el nodo 5 (r = 0.0643)
Revisando el nodo 5 con tabú, obtuvimos que éste es aceptado
Hemos aceptado el nodo. Ahora tenemos el camino siguiente
[1] [2] [7] [3] [6] [5] [0]
[0] [0] [0] [1] [0] [0] [0]

Escogeremos el nodo 6
Tenemos las probabilidades individuales y acumuladas de que de 5 se vaya a
1: [0.0000] [0.0000]
2: [0.0000] [0.0000]
3: [0.0167] [0.0167]
4: [0.0200] [0.0367]
5: [0.0000] [0.0367]
6: [0.0200] [0.0567]
7: [0.0000] [0.0567]

```

```

Hemos escogido el nodo 3 (r = 0.0048)
Revisando el nodo 3 con tabú, obtuvimos que éste no es aceptado
Tendremos que repetir la selección aleatoria por posible encrucijada (La hormiga lleva 1 repeticiones)
Tenemos las probabilidades individuales y acumuladas de que de 5 se vaya a
1: [0.0000] [0.0000]
2: [0.0000] [0.0000]
3: [0.0167] [0.0167]
4: [0.0200] [0.0367]
5: [0.0000] [0.0367]
6: [0.0200] [0.0567]
7: [0.0000] [0.0567]

Hemos escogido el nodo 6 (r = 0.0424)
Revisando el nodo 6 con tabú, obtuvimos que éste no es aceptado
Tendremos que repetir la selección aleatoria por posible encrucijada (La hormiga lleva 2 repeticiones)
Tenemos las probabilidades individuales y acumuladas de que de 5 se vaya a
1: [0.0000] [0.0000]
2: [0.0000] [0.0000]
3: [0.0167] [0.0167]
4: [0.0200] [0.0367]
5: [0.0000] [0.0367]
6: [0.0200] [0.0567]
7: [0.0000] [0.0567]

Hemos escogido el nodo 4 (r = 0.0325)
Revisando el nodo 4 con tabú, obtuvimos que éste es aceptado
Hemos aceptado el nodo. Ahora tenemos el camino siguiente
[1] [2] [7] [3] [6] [5] [4]
[0] [0] [0] [0] [0] [0] [0]
Hemos llegado al nodo objetivo

La hormiga produjo un recorrido de L = 26

```

## Conclusiones:

Creo que evidentemente sí se necesitaron de elementos avanzados de modelación para trabajar con esta práctica, realmente eran muchas estructuras de datos las que se necesitaban tomar en cuenta y por ello se debía analizar con mayor detenimiento de dónde tomar los datos, qué datos tomar, cómo interpretar cada uno, entre otras cosas. Podemos observar que la colonia funciona correctamente y se encuentra generalizada a problemas de grafos, hormigas e iteraciones arbitrarias por lo que fue una interesante forma de ver la aplicación de la metaheurística.

## Referencias:

- Purcell, E. (2007). *Cálculo*. Londres: Pearson Education.
- Talbi, E. (2009). *Metaheuristics from design to implementation*. New Jersey: John Wiley & Sons Publication.
- Torres, A. (2020). *Apuntes: Optimización Inteligente*. 5° ICI. México: Universidad Autónoma de Aguascalientes.

## **Anexos:**

### **Anexo 1: Código del programa en lenguaje C:**

```
/*
    Universidad Autónoma de Aguascalientes

        Centro de Ciencias Básicas
    Departamento de Ciencias de la Computación
        Optimización Inteligente

                5° "A"

        Práctica 7: Colonia de Hormigas

        Doctora Aurora Torres Soto

        Alumno: Joel Alejandro Espinoza Sánchez

        Fecha de Entrega: 14 de noviembre del 2020

Descripción:
*/
//Cargamos las librerías
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
#include <time.h>
#include <math.h>

void setValues1(float alpha, int autom, float beta, float Q, int
qh, int qv, float rho, float tau, float *alphaP, int *automP,
float *betaP, float *QP, int *qhP, int *qvP, float *rhoP, float
*tauP);
void fillMatrix1D(int a, float matrix[a], float n);
void fillMatrix2D(int a, int b, float matrix[a][b], float n);
void fillMatrix3D(int a, int b, int c, float matrix[a][b][c],
float n);
void setValues2(int qv, int qh, float graph[qv][qv], float
pheromone[qv][qv], float vision[qv][qv], float beginEnd[qh][2]);
void startProcess(int i, int qh, int qv, float tabu[qh][qv], float
beginEnd[qh][2], float availableV[qv]);
```



```

int acceptVertex(int i, int qh, int qv, int value, float
tabu[qh][qv]);
int getL(int i, int qh, int qv, float graph[qv][qv], float
tabu[qh][qv], float beginEnd[qh][2]);

main()
{
    setlocale(LC_ALL, "");
    srand(time(NULL));

    //1. Declaramos las variables que usaremos
    /*
        qv: Cantidad de vértices del grafo
        qh: Cantidad de hormigas
        tau: Valor de la feromona inicial
        Q: Parámetro
        alpha: Parámetro
        beta: Parámetro
        rho: Parámetro
        autom: Modo de acción (Para autom >=1 se tomará el
número de autom como las repeticiones)
        repeat: Sirve para repetir todo el código nuevamente
        repeat1: Sirve para repetir la colonia en modo manual
        r: Número aleatorio
        accepted: Bandera de aceptación de vértice
        checkIfStuck: Iterador que no tolerará un número
determinado de iteraciones. Si se alcanzan, se interpretará que la
hormiga se encerró
        L: Longitud del recorrido completo realizado por la
hormiga
    */
    int qv, qh, autom, repeat, repeat1, repeat2, accepted,
checkIfStuck, L, h, i, j, k, ip, jp, kp;
    float tau, Q, alpha, beta, rho, r;

    printf("===== COLONIA DE HORMIGAS
=====\\n");

    do
    {
        alpha = 1;
        autom = 100;

```

```

    beta = 1;
    Q = 1;
    qh = 1;
    qv = 7;
    repeat = 0;
    rho = 0.01;
    tau = 0.1;

    //2. Calibramos los primeros valores del programa

    setValues1(alpha, autom, beta, Q, qh, qv, rho, tau, &alpha, &autom, &beta, &Q, &qh, &qv, &rho, &tau);

    //3. Declaramos más variables
    /*
        graph: La matriz de adyacencia del grafo que
analizaremos
        INT
        pheromone: La matriz de feromonas

        FLOAT
        vision: La matriz de visibilidad

        FLOAT
        tabu: La lista tabú

        INT
        beginEnd: Matriz con los valores de inicio y fin
de cada hormiga
        INT
        availableV: Vector donde cada elemento representa
cada vértice del grafo (1 está disponible o 0 no lo está)
        INT
        prob: Probabilidad individual y acumulada de tomar
cada nodo
        FLOAT
        dtau: La diferencia de tau que hay por hormiga en
cada nodo
        FLOAT
    */

```

```
float graph[qv][qv], pheromone[qv][qv], vision[qv][qv],
tabu[qh][qv], beginEnd[qh][2], availableV[qv], prob[qv][2],
dtau[qh][qv][qv];
```

```
//4. Inicializamos las estructuras
```

```
//Limpiamos graph
```

```
fillMatrix2D(qv,qv,graph,0);
```

```
//Inicializamos pheromone
```

```
fillMatrix2D(qv,qv,pheromone,tau);
```

```
//Limpiamos vision
```

```
fillMatrix2D(qv,qv,vision,0);
```

```
//Limpiamos tabu
```

```
fillMatrix2D(qh,qv,tabu,0);
```

```
//Limpiamos beginEnd
```

```
fillMatrix2D(qh,2,beginEnd,0);
```

```
//Inicializamos availableV
```

```
fillMatrix1D(qv,availableV,1);
```

```
//Limpiamos prob
```

```
fillMatrix2D(qv,2,prob,0);
```

```
//5. Calibramos los demás valores del programa
```

```
setValues2(qv,qh,graph,pheromone,vision,beginEnd);
```

```
//Inicializamos vision
```

```
for(i = 0; i < qv; i++)
```

```
{
```

```
    for(j = 0; j < qv; j++)
```

```
    {
```

```
        if(graph[i][j] == 0)
```

```
        {
```

```
            vision[i][j] = 0;
```

```
        }
```

```
    else
```

```
    {
```

```
        vision[i][j] =
```

```
(pow(pheromone[i][j],alpha))/(pow(graph[i][j],beta));
```

```

    }
}

printf("\n\n\n\n");

if(autom == 0)
{
    printf("El grafo a evaluar es:\n");
    for(ip = 0; ip < qv; ip++)
    {
        for(jp = 0; jp < qv; jp++)
        {
            printf("[%d] ", (int)graph[ip][jp]);
        }
        printf("\n");
    }
    printf("\n");

    printf("La matriz de feromonas antes del comienzo
del algoritmo es:\n");
    for(ip = 0; ip < qv; ip++)
    {
        for(jp = 0; jp < qv; jp++)
        {
            printf("[%4f] ", pheromone[ip][jp]);
        }
        printf("\n");
    }
    printf("\n");

    printf("La matriz de visibilidad, entonces,
es:\n");
    for(ip = 0; ip < qv; ip++)
    {
        for(jp = 0; jp < qv; jp++)
        {
            printf("[%4f] ", vision[ip][jp]);
        }
        printf("\n");
    }
    printf("\n");
}

```

```

        getchar();
        getchar();
    }

    //6. Comenzamos la colonia de hormigas, la cual se
    repetirá tantas veces como se haya calibrado autom (si es 0, se
    decide el paro)
    h = 0;
    repeat2 = 1;
    do
    {
        //Inicializamos dtau
        fillMatrix3D(qh,qv,qv,dtau,0);

        if(autom == 0)
        {
            printf("\n");
            printf("\n");
            printf("Comenzamos la %dº iteración\n\n", h +
1);
        }

        //El procedimiento de la colonia se hará para cada
hormiga
        i = 0;
        do
        {
            if(autom == 0)
            {
                printf(" Comenzamos a trabajar con la
hormiga %d\n", i + 1);
                getchar();
            }

            //Inicializamos tabu y availableV con los
valores proporcionados por beginEnd

            startProcess(i,qh,qv,tabu,beginEnd,availableV);

            if(autom == 0)
            {

```

```

        printf("    Inicializamos la lista tabú
con el valor del nodo de inicio:\n    ");
        for(ip = 0; ip < qh; ip++)
        {
            for(jp = 0; jp < qv; jp++)
            {
                printf("[%d] ",
(int)tabu[ip][jp]);

            }
            printf("\n");
        }

```

```

        printf("    Inicializamos la lista de
nodos disponibles:\n    ");
        for(ip = 0; ip < qv; ip++)
        {
            printf("[%d] ",
(int)availableV[ip]);

        }
        printf("\n");

        getchar();
    }

```

//Todavía, el procedimiento de selección de camino de cada hormiga tiene que hacerse por cada nodo como máximo  $qv - 1$  veces

```

        j = 0;
        do
        {
            if(autom == 0)
            {
                printf("\n");
                printf("    Escogeremos el nodo
%d\n", j + 1);

            }

```

```

        //Debemos repetir este análisis hasta
que el nodo haya sido reconocido como válido
        accepted = 0;
        checkIfStuck = 0;
        do

```

```

{
    //A partir del nodo otorgado,
    vamos al renglón en vision con el que rellenaremos prob y
    procedemos a rellenar prob
    prob[0][0] =
vision[(int)tabu[i][j] - 1][0];
    prob[0][1] = prob[0][0];
    for(k = 1; k < qv; k++)
    {
        prob[k][0] =
vision[(int)tabu[i][j] - 1][k];
        prob[k][1] = prob[k][0] +
prob[k - 1][1];
    }

    //Generamos un número aleatorio y
    discretizamos el valor entre 10000 posibilidades, todas dentro del
    rango
    r = rand() % 10000;
    r = r/(10000/prob[qv - 1][1]);

    if(autom == 0)
    {
        printf("          Tenemos las
probabilidades individuales y acumuladas de que de %d se vaya
a\n",(int)tabu[i][j]);
        for(ip = 0; ip < qv; ip++)
        {
            printf("          %d:
", ip + 1);
            for(jp = 0; jp < 2;
jp++)
            {
                printf("[%.4f]
", prob[ip][jp]);
            }
            printf("\n");
        }
        printf("\n");
    }

    //Ubicaremos el nodo seleccionado

```

```

for(k = 0; k < qv; k++)
{
    if(r < prob[k][1])
    {
        //Encontramos que cayó
        break;
    }
}

if(autom == 0)
{
    printf("      Hemos escogido
el nodo %d (r = %.4f)\n",k + 1,r);
}

//Ahora toca validar si ese men
está disponible o no
accepted =
acceptVertex(i,qh,qv,k+1,tabu);

if(autom == 0)
{
    if(accepted == 1)
    {
        printf("      Revisando
el nodo %d con tabú, obtuvimos que éste es aceptado\n", k + 1);
    }
    else
    {
        printf("      Revisando
el nodo %d con tabú, obtuvimos que éste no es aceptado\n", k + 1);
    }
}

//También tenemos que validar si
la hormiga no se quedó encerrada
if(accepted == 0)
{
    checkIfStuck++;

    if(autom == 0)

```



```

        {
            printf("      Tendremos
que repetir la selección aleatoria por posible encrucijada (La
hormiga lleva %d repeticiones)\n", checkIfStuck);
        }
    }

    if(checkIfStuck == 100)
    {
        if(autom == 0)
        {
            printf("      Hemos
interpretado que la hormiga se quedó atorada\n Reiniciaremos el
proceso completo\n");
        }

        startProcess(i,qh,qv,tabu,beginEnd,availableV);
        checkIfStuck = 0;
        j = 0;
    }
}
while(accepted == 0);

//Hemos aceptado el nodo, vamos a
actualizar la lista tabú y availableV
j++;
tabu[i][j] = k + 1;
availableV[(int)tabu[i][j] - 1] = 0;

if(autom == 0)
{
    printf("      Hemos aceptado el
nodo. Ahora tenemos el camino siguiente\n      ");
    for(ip = 0; ip < qh; ip++)
    {
        for(jp = 0; jp < qv; jp++)
        {
            printf("[%d] ",
(int)tabu[ip][jp]);

        }
        printf("\n      ");
    }
}

```

```

    }
    for(ip = 0; ip < qv; ip++)
    {
        printf("[%d] ",
(int)availableV[ip]);
    }
    printf("\n");
}

//Chequemos si el nodo aceptado es el
definido como el último. Así la hormiga se sentirá realizada
if(k + 1 == beginEnd[i][1])
{
    if(autom == 0)
    {
        printf("        Hemos llegado
al nodo objetivo\n");
    }
    break;
}
}
while(j < qv - 1);

L = getL(i,qh,qv,graph,tabu,beginEnd);

if(autom == 0)
{
    printf("\n");
    printf("        La hormiga produjo un
recorrido de L = %d\n",L);
    getchar();
}

if(autom != 0)
{
    printf("En la iteración %d, la hormiga
%d produjo el camino:\n",h + 1, i + 1);
    for(jp = 0; jp < qv; jp++)
    {
        printf("[%d] ",
(int)tabu[ip][jp]);
    }
}

```

```

        printf("\n");
        printf("Con un recorrido de valor
%d\n",L);
    }

    for(j = 0; j < qv; j++)
    {
        if((int)tabu[i][j] ==
(int)beginEnd[i][1])
        {
            break;
        }
        else
        {
            dtau[i][(int)tabu[i][j] -
1][(int)tabu[i][j + 1] - 1] = Q/L;
        }
    }

    if(autom == 0)
    {
        printf("    Se ha generado el siguiente
incremento de feromonas:\n    ");
        for(jp = 0; jp < qv; jp++)
        {
            for(kp = 0; kp < qv; kp++)
            {
                printf("[%.4f] ",
dtau[i][jp][kp]);
            }
            printf("\n    ");
        }
        getchar();
    }

    i++;
}
while(i < qh);

//Las hormigas han llegado al nodo destino, ahora
procedemos a actualizar pheromone
for(j = 0; j < qv; j++)

```

```

        {
            for(k = 0; k < qv; k++)
            {
                for(i = 0; i < qh; i++)
                {
                    dtau[0][j][k] = dtau[0][j][k] +
dtau[i][j][k];
                }
                pheromone[j][k] = ((1-
rho)*(pheromone[j][k])) + dtau[0][j][k];
            }
        }

        if(autom == 0)
        {
            printf("    Tras el incremento y evaporación
de feromonas, la nueva matriz de feromonas es:\n    ");
            for(jp = 0; jp < qv; jp++)
            {
                for(kp = 0; kp < qv; kp++)
                {
                    printf("[%.4f] ",
pheromone[jp][kp]);
                }
                printf("\n    ");
            }
            getchar();
        }
    }

```

//La colonia de hormigas ha finalizado,  
preguntamos por una nueva iteración o terminar el programa, según  
sea el caso

```

        if(autom == 0)
        {
            printf("¿Desea usar un nuevo grupo de
hormigas?\n");

            printf("0. No\n");
            printf("1. Sí\n");
            scanf("%d",&repeat1);
            if(repeat1 == 0)

```

```

        {
            repeat2 = 0;
        }
        if(repeat1 == 1)
        {
            repeat2 = 1;
        }
        h++;
    }
    else
    {
        h++;
        if(h == autom)
        {
            repeat2 = 0;
        }
    }
}
while(repeat2 == 1);

printf("===== Reporte Final
=====\\n");
printf("Hemos analizado el grafo:\\n");
for(ip = 0; ip < qv; ip++)
{
    for(jp = 0; jp < qv; jp++)
    {
        printf("[%d] ", (int)graph[ip][jp]);
    }
    printf("\\n");
}
printf("\\n");

printf("La matriz de feromonas resultante es:\\n");
for(ip = 0; ip < qv; ip++)
{
    for(jp = 0; jp < qv; jp++)
    {
        printf("[%4f] ", pheromone[ip][jp]);
    }
    printf("\\n");
}

```

```

printf("\n");

printf("La matriz de visibilidad resultante es:\n");
for(ip = 0; ip < qv; ip++)
{
    for(jp = 0; jp < qv; jp++)
    {
        printf("[%.4f] ", vision[ip][jp]);
    }
    printf("\n");
}
printf("\n");

printf("\n");
printf("\n");
printf("\n");
printf("¿Desea repetir el código?\n");
printf("0. No\n");
printf("1. Sí\n");
scanf("%d",&repeat);
}
while(repeat == 1);

getchar();
}

void setValues1(float alpha, int autom, float beta, float Q, int
qh, int qv, float rho, float tau, float *alphaP, int *automP,
float *betaP, float *QP, int *qhP, int *qvP, float *rhoP, float
*tauP)
{
    int aux, done = 0;
    *alphaP = alpha;
    *automP = autom;
    *betaP = beta;
    *QP = Q;
    *qhP = qh;
    *qvP = qv;
    *rhoP = rho;

```

```

*tauiP = tau;

do
{
    printf("\n");
    printf("-----\n");
    printf("|  Calibración del algoritmo:\n");
    printf("|  Seleccione algún número si desea cambiar su
valor (o 0 para continuar):\n");
    printf("|  0: Listo. Continuar\n");
    printf("|  1: tau (Tau inicial): %.4f\n",tau);
    printf("|  2: alpha (a): %.4f\n",alpha);
    printf("|  3: beta (a): %.4f\n",beta);
    printf("|  4: rho (a): %.4f\n",rho);
    printf("|  5: Q (a): %.4f\n",Q);
    printf("|  6: qh (Cantidad de hormigas): %d\n",qh);
    printf("|  7: qv (Cantidad de vértices que posee el
grafo): %d\n",qv);
    printf("|  8: autom (Modo de acción): %d\n",autom);
    printf("|  ");
    scanf("%d",&aux);

    switch (aux)
    {
        case 0:
        {
            //Se continúa con el programa
            done = 1;
            break;
        }
        case 1:
        {
            //Se modifica tau

            printf("|      Inserte un nuevo valor
para tau (Antiguo valor para tau: T0 = %.4f)\n",tau);
            printf("|  ");
            scanf("%f",&tau);
            *tauP = tau;

            break;
        }
    }
}

```

```

case 2:
{
    //Se modifica alpha

    printf("|      Inserte un nuevo valor
para alpha (Antiguo valor para alpha: alpha = %.4f)\n",alpha);
    printf("|  ");
    scanf("%f",&alpha);
    *alphaP = alpha;

    break;
}
case 3:
{
    //Se modifica beta

    printf("|      Inserte un nuevo valor
para alpha (Antiguo valor para beta: beta = %.4f)\n",beta);
    printf("|  ");
    scanf("%f",&beta);
    *betaP = beta;

    break;
}
case 4:
{
    //Se modifica rho

    printf("|      Inserte un nuevo valor
para rho (Antiguo valor para rho: rho = %.4f)\n",rho);
    printf("|  ");
    scanf("%f",&rho);
    *rhoP = rho;

    break;
}
case 5:
{
    //Se modifica Q

    printf("|      Inserte un nuevo valor
para K (Antiguo valor para Q: Q = %.4f)\n",Q);

```



```

        printf("|  ");
        scanf("%f",&Q);
        *QP = Q;

        break;
    }
    case 6:
    {
        //Se modifica qh

        printf("|      Inserte un nuevo valor
para qh (Antiguo valor para qh: qh = %d)\n",qh);
        printf("|  ");
        scanf("%d",&qh);
        *qhP = qh;

        break;
    }
    case 7:
    {
        //Se modifica qv

        printf("|      Inserte un nuevo valor
para qv (Antiguo valor para qv: qv = %d)\n",qv);
        printf("|  ");
        scanf("%d",&qv);
        *qvP = qv;

        break;
    }
    case 8:
    {
        //Se modifica autom

        printf("|      Inserte un nuevo valor
para autom (Antiguo valor para autom: autom = %d)\n",autom);
        printf("|  ");
        scanf("%d",&autom);
        *automP = autom;

        break;
    }

```

```

        default:
        {
            //Valor no válido

            printf("|      Ha insertado un número
inválido\n");

            break;
        }
    }
}
while(done == 0);

return;
}

void fillMatrix1D(int a, float matrix[a], float n)
{
    int i,j;
    for(i = 0; i < a; i++)
    {
        matrix[i] = n;
    }
    return;
}

void fillMatrix2D(int a, int b, float matrix[a][b], float n)
{
    int i,j;
    for(i = 0; i < a; i++)
    {
        for(j = 0; j < b; j++)
        {
            matrix[i][j] = n;
        }
    }
    return;
}

void fillMatrix3D(int a, int b, int c, float matrix[a][b][c],
float n)
{

```

```

int i,j,k;
for(i = 0; i < a; i++)
{
    for(j = 0; j < b; j++)
    {
        for(k = 0; k < c; k++)
        {
            matrix[i][j][k] = n;
        }
    }
}
return;
}

void setValues2(int qv, int qh, float graph[qv][qv], float
pheromone[qv][qv], float vision[qv][qv], float beginEnd[qh][2])
{
    int aux, done = 0,i,j;
    do
    {
        printf("\n");
        printf("-----\n");
        printf("|  Calibración del algoritmo:\n");
        printf("|  Seleccione algún número si desea cambiar su
valor (o 0 para continuar):\n");
        printf("|  0: Listo. Continuar\n");
        printf("|  1: graph (El grafo a evaluar):\n");
        for(i = 0; i < qv; i++)
        {
            printf("|      ");
            for(j = 0; j < qv; j++)
            {
                printf("[%d] ",(int)graph[i][j]);
            }
            printf("\n");
        }
        printf("|  2: BeginEnd (Los puntos de inicio y final de
cada hormiga):\n");
        for(i = 0; i < qh; i++)
        {
            printf("|      ");
            for(j = 0; j < 2; j++)

```

```

        {
            printf("[%d] ",(int)beginEnd[i][j]);
        }
        printf("\n");
    }
    printf("|  ");
    scanf("%d",&aux);

    switch (aux)
    {
        case 0:
        {
            //Se continúa con el programa
            done = 1;
            break;
        }
        case 1:
        {
            //Se modifica graph

            for(i = 0; i < qv; i++)
            {
                for(j = 0; j < qv; j++)
                {
                    printf("|      Inserte el peso
de la arista que une al vértice %d con el vértice %d (0 para
indicar que no existe tal unión)\n",i + 1, j + 1);
                    printf("|  ");
                    scanf("%f",&graph[i][j]);
                }
            }

            break;
        }
        case 2:
        {
            //Se modifican los puntos de arranque y
fin de cada hormiga (tabu)

            for(i = 0; i < qh; i++)
            {

```

```

        printf("|      Inserte el nodo
inicial de la hormiga %d:\n",i + 1);
        printf("|  ");
        scanf("%f",&beginEnd[i][0]);
        printf("|      Inserte el nodo
final de la hormiga %d:\n",i + 1);
        printf("|  ");
        scanf("%f",&beginEnd[i][1]);
    }

    break;
}

}

while(done == 0);
}

void startProcess(int i, int qh, int qv, float tabu[qh][qv], float
beginEnd[qh][2], float availableV[qv])
{
    int j;

    for(j = 0; j < qv; j++)
    {
        tabu[i][j] = 0;
        availableV[j] = 1;
    }

    tabu[i][0] = (int)beginEnd[i][0];
    availableV[(int)tabu[i][0] - 1] = 0;

    return;
}

int acceptVertex(int i, int qh, int qv, int value, float
tabu[qh][qv])
{
    int j;
    for(j = 0; j < qv; j++)
    {
        if((int)tabu[i][j] == value)
        {

```

```

        return 0;
    }
}
return 1;
}

int getL(int i, int qh, int qv, float graph[qv][qv], float
tabu[qh][qv], float beginEnd[qh][2])
{
    int j, L = 0;
    for(j = 0; j < qv; j++)
    {
        if((int)tabu[i][j] == (int)beginEnd[i][1])
        {
            return L;
        }
        else
        {
            L = L + graph[(int)tabu[i][j] - 1][(int)tabu[i][j]
+ 1] - 1];
        }
    }
}

```

## Anexo 2: Impresión de pantalla completa al ejecutar el programa

### Modo de acción 0 (Manual)

```
===== COLONIA DE HORMIGAS =====
-----
Calibración del algoritmo:
Seleccione algún número si desea cambiar su valor (o 0 para continuar):
0: Listo. Continuar
1: tau1 (Tau inicial): 0.1000
2: alpha (a): 1.0000
3: beta (a): 1.0000
4: rho (a): 0.0100
5: Q (a): 1.0000
6: qh (Cantidad de hormigas): 1
7: qv (Cantidad de vértices que posee el grafo): 7
8: autom (Modo de acción): 0
0 1 0 5 3 0 0 6 0 5 0 4 0 0 0 5 3 4 0 0 6 3 3 0 0 0 0 5 0 4 0 0 6 5 0 5 0 6 0 3 0 5 0 0 0 5 3 4 0 0 0 2 1 4 0
-----
Calibración del algoritmo:
Seleccione algún número si desea cambiar su valor (o 0 para continuar):
0: Listo. Continuar
1: graph (El grafo a evaluar):
  [0] [0] [0] [0] [0] [0] [0]
  [0] [0] [0] [0] [0] [0] [0]
  [0] [0] [0] [0] [0] [0] [0]
  [0] [0] [0] [0] [0] [0] [0]
  [0] [0] [0] [0] [0] [0] [0]
  [0] [0] [0] [0] [0] [0] [0]
  [0] [0] [0] [0] [0] [0] [0]
  [0] [0] [0] [0] [0] [0] [0]
2: BeginEnd (Los puntos de inicio y final de cada hormiga):
  [0] [0]
```

```
El grafo a evaluar es:
[0] [5] [3] [0] [0] [6] [0]
[5] [0] [4] [0] [0] [0] [5]
[3] [4] [0] [0] [6] [3] [3]
[0] [0] [0] [0] [5] [0] [4]
[0] [0] [6] [5] [0] [5] [0]
[6] [0] [3] [0] [5] [0] [0]
[0] [5] [3] [4] [0] [0] [0]

La matriz de feromonas antes del comienzo del algoritmo es:
[0.1000] [0.1000] [0.1000] [0.1000] [0.1000] [0.1000] [0.1000]
[0.1000] [0.1000] [0.1000] [0.1000] [0.1000] [0.1000] [0.1000]
[0.1000] [0.1000] [0.1000] [0.1000] [0.1000] [0.1000] [0.1000]
[0.1000] [0.1000] [0.1000] [0.1000] [0.1000] [0.1000] [0.1000]
[0.1000] [0.1000] [0.1000] [0.1000] [0.1000] [0.1000] [0.1000]
[0.1000] [0.1000] [0.1000] [0.1000] [0.1000] [0.1000] [0.1000]
[0.1000] [0.1000] [0.1000] [0.1000] [0.1000] [0.1000] [0.1000]

La matriz de visibilidad, entonces, es:
[0.0000] [0.0200] [0.0333] [0.0000] [0.0000] [0.0167] [0.0000]
[0.0200] [0.0000] [0.0250] [0.0000] [0.0000] [0.0000] [0.0200]
[0.0333] [0.0250] [0.0000] [0.0000] [0.0167] [0.0333] [0.0333]
[0.0000] [0.0000] [0.0000] [0.0000] [0.0200] [0.0000] [0.0250]
[0.0000] [0.0000] [0.0167] [0.0200] [0.0000] [0.0200] [0.0000]
[0.0167] [0.0000] [0.0333] [0.0000] [0.0200] [0.0000] [0.0000]
[0.0000] [0.0200] [0.0333] [0.0250] [0.0000] [0.0000] [0.0000]
```

Comenzamos la 1º iteración

Comenzamos a trabajar con la hormiga 1

Inicializamos la lista tabú con el valor del nodo de inicio:

[1] [0] [0] [0] [0] [0] [0]

Inicializamos la lista de nodos disponibles:

[0] [1] [1] [1] [1] [1] [1]

Escogeremos el nodo 1

Tenemos las probabilidades individuales y acumuladas de que de 1 se vaya a

1: [0.0000] [0.0000]

2: [0.0200] [0.0200]

3: [0.0333] [0.0533]

4: [0.0000] [0.0533]

5: [0.0000] [0.0533]

6: [0.0167] [0.0700]

7: [0.0000] [0.0700]

Hemos escogido el nodo 3 (r = 0.0245)

Revisando el nodo 3 con tabú, obtuvimos que éste es aceptado

Hemos aceptado el nodo. Ahora tenemos el camino siguiente

[1] [3] [0] [0] [0] [0] [0]

[0] [1] [0] [1] [1] [1] [1]

Escogeremos el nodo 2

Tenemos las probabilidades individuales y acumuladas de que de 3 se vaya a

1: [0.0333] [0.0333]

2: [0.0250] [0.0583]

3: [0.0000] [0.0583]

4: [0.0000] [0.0583]

5: [0.0167] [0.0750]

6: [0.0333] [0.1083]

7: [0.0333] [0.1417]

Hemos escogido el nodo 2 (r = 0.0418)

Revisando el nodo 2 con tabú, obtuvimos que éste es aceptado

Hemos aceptado el nodo. Ahora tenemos el camino siguiente

[1] [3] [2] [0] [0] [0] [0]

[0] [0] [0] [1] [1] [1] [1]

Escogeremos el nodo 3

Tenemos las probabilidades individuales y acumuladas de que de 2 se vaya a

1: [0.0200] [0.0200]

2: [0.0000] [0.0200]

3: [0.0250] [0.0450]

4: [0.0000] [0.0450]

5: [0.0000] [0.0450]

6: [0.0000] [0.0450]

7: [0.0200] [0.0650]



```

Hemos escogido el nodo 1 ( $r = 0.0110$ )
Revisando el nodo 1 con tabú, obtuvimos que éste no es aceptado
Tendremos que repetir la selección aleatoria por posible encrucijada (La hormiga lleva 6 repeticiones)
Tenemos las probabilidades individuales y acumuladas de que de 2 se vaya a
1: [0.0200] [0.0200]
2: [0.0000] [0.0200]
3: [0.0250] [0.0450]
4: [0.0000] [0.0450]
5: [0.0000] [0.0450]
6: [0.0000] [0.0450]
7: [0.0200] [0.0650]

Hemos escogido el nodo 7 ( $r = 0.0507$ )
Revisando el nodo 7 con tabú, obtuvimos que éste es aceptado
Hemos aceptado el nodo. Ahora tenemos el camino siguiente
[1] [3] [2] [7] [0] [0] [0]
[0] [0] [0] [1] [1] [1] [0]

Escogeremos el nodo 4
Tenemos las probabilidades individuales y acumuladas de que de 7 se vaya a
1: [0.0000] [0.0000]
2: [0.0200] [0.0200]
3: [0.0333] [0.0533]
4: [0.0250] [0.0783]
5: [0.0000] [0.0783]
6: [0.0000] [0.0783]
7: [0.0000] [0.0783]

Hemos escogido el nodo 3 ( $r = 0.0390$ )
Revisando el nodo 3 con tabú, obtuvimos que éste no es aceptado
Tendremos que repetir la selección aleatoria por posible encrucijada (La hormiga lleva 1 repeticiones)
Tenemos las probabilidades individuales y acumuladas de que de 7 se vaya a
1: [0.0000] [0.0000]
2: [0.0200] [0.0200]
3: [0.0333] [0.0533]
4: [0.0250] [0.0783]
5: [0.0000] [0.0783]
6: [0.0000] [0.0783]
7: [0.0000] [0.0783]

Hemos escogido el nodo 3 ( $r = 0.0427$ )
Revisando el nodo 3 con tabú, obtuvimos que éste no es aceptado
Tendremos que repetir la selección aleatoria por posible encrucijada (La hormiga lleva 2 repeticiones)
Tenemos las probabilidades individuales y acumuladas de que de 7 se vaya a
1: [0.0000] [0.0000]
2: [0.0200] [0.0200]
3: [0.0333] [0.0533]
4: [0.0250] [0.0783]
5: [0.0000] [0.0783]
6: [0.0000] [0.0783]
7: [0.0000] [0.0783]

Hemos escogido el nodo 4 ( $r = 0.0652$ )
Revisando el nodo 4 con tabú, obtuvimos que éste es aceptado
Hemos aceptado el nodo. Ahora tenemos el camino siguiente
[1] [3] [2] [7] [4] [0] [0]
[0] [0] [0] [0] [1] [1] [0]
Hemos llegado al nodo objetivo

```

La hormiga produjo un recorrido de  $L = 16$

```

Se ha generado el siguiente incremento de feromonas:
[0.0000] [0.0000] [0.0625] [0.0000] [0.0000] [0.0000] [0.0000]
[0.0000] [0.0000] [0.0000] [0.0000] [0.0000] [0.0000] [0.0625]
[0.0000] [0.0625] [0.0000] [0.0000] [0.0000] [0.0000] [0.0000]
[0.0000] [0.0000] [0.0000] [0.0000] [0.0000] [0.0000] [0.0000]
[0.0000] [0.0000] [0.0000] [0.0000] [0.0000] [0.0000] [0.0000]
[0.0000] [0.0000] [0.0000] [0.0000] [0.0000] [0.0000] [0.0000]
[0.0000] [0.0000] [0.0000] [0.0625] [0.0000] [0.0000] [0.0000]

Tras el incremento y evaporación de feromonas, la nueva matriz de feromonas es:
[0.0990] [0.0990] [0.2240] [0.0990] [0.0990] [0.0990] [0.0990]
[0.0990] [0.0990] [0.0990] [0.0990] [0.0990] [0.0990] [0.2240]
[0.0990] [0.2240] [0.0990] [0.0990] [0.0990] [0.0990] [0.0990]
[0.0990] [0.0990] [0.0990] [0.0990] [0.0990] [0.0990] [0.0990]
[0.0990] [0.0990] [0.0990] [0.0990] [0.0990] [0.0990] [0.0990]
[0.0990] [0.0990] [0.0990] [0.0990] [0.0990] [0.0990] [0.0990]
[0.0990] [0.0990] [0.0990] [0.2240] [0.0990] [0.0990] [0.0990]

¿Desea usar un nuevo grupo de hormigas?
0. No
1. Sí

```

Modo de acción 1 o superior (Automático)

```

-----
| Calibración del algoritmo:
| Seleccione algún número si desea cambiar su valor (o 0 para continuar):
| 0: Listo. Continuar
| 1: tau1 (Tau inicial): 0.1000
| 2: alpha (a): 1.0000
| 3: beta (a): 1.0000
| 4: rho (a): 0.0100
| 5: Q (a): 1.0000
| 6: qh (Cantidad de hormigas): 1
| 7: qv (Cantidad de vértices que posee el grafo): 7
| 8: autom (Modo de acción): 8
| 0 1 0 5 3 0 0 6 0 5 0 4 0 0 0 5 3 4 0 0 6 3 3 0 0 0 0 5 0 4 0 0 6 5 0 5 0 6 0 3 0 5 0 0 0 5 3 4 0 0 0 2 1 4 0
|
|-----
| Calibración del algoritmo:
| Seleccione algún número si desea cambiar su valor (o 0 para continuar):
| 0: Listo. Continuar
| 1: graph (El grafo a evaluar):
| [0] [0] [0] [0] [0] [0] [0] [0]
| [0] [0] [0] [0] [0] [0] [0] [0]
| [0] [0] [0] [0] [0] [0] [0] [0]
| [0] [0] [0] [0] [0] [0] [0] [0]
| [0] [0] [0] [0] [0] [0] [0] [0]
| [0] [0] [0] [0] [0] [0] [0] [0]
| [0] [0] [0] [0] [0] [0] [0] [0]
| 2: BeginEnd (Los puntos de inicio y final de cada hormiga):
| [0] [0]

```

```

En la iteración 1, la hormiga 1 produjo el camino:
[1] [3] [6] [5] [4] [0] [0]
Con un recorrido de valor 16
En la iteración 2, la hormiga 1 produjo el camino:
[1] [3] [6] [5] [4] [0] [0]
Con un recorrido de valor 16
En la iteración 3, la hormiga 1 produjo el camino:
[1] [2] [3] [7] [4] [0] [0]
Con un recorrido de valor 16
En la iteración 4, la hormiga 1 produjo el camino:
[1] [2] [3] [7] [4] [0] [0]
Con un recorrido de valor 16
En la iteración 5, la hormiga 1 produjo el camino:
[1] [2] [3] [6] [5] [4] [0]
Con un recorrido de valor 22
En la iteración 6, la hormiga 1 produjo el camino:
[1] [6] [3] [5] [4] [0] [0]
Con un recorrido de valor 20
En la iteración 7, la hormiga 1 produjo el camino:
[1] [3] [7] [4] [0] [0] [0]
Con un recorrido de valor 10
En la iteración 8, la hormiga 1 produjo el camino:
[1] [3] [2] [7] [4] [0] [0]
Con un recorrido de valor 16
===== Reporte Final =====
Hemos analizado el grafo:
[0] [5] [3] [0] [0] [6] [0]
[5] [0] [4] [0] [0] [0] [5]
[3] [4] [0] [0] [6] [3] [3]
[0] [0] [0] [0] [5] [0] [4]
[0] [0] [6] [5] [0] [5] [0]
[6] [0] [3] [0] [5] [0] [0]
[0] [5] [3] [4] [0] [0] [0]

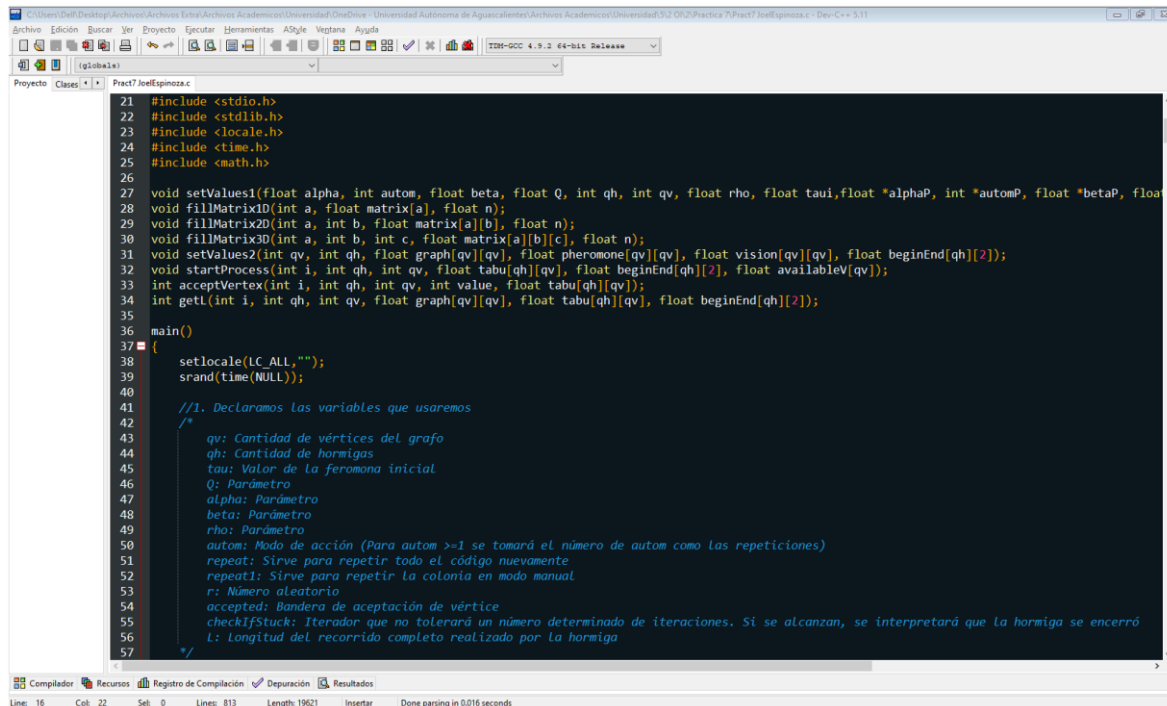
La matriz de feromonas resultante es:
[0.0923] [0.4194] [0.6495] [0.0923] [0.0923] [0.1903] [0.0923]
[0.0923] [0.0923] [0.4194] [0.0923] [0.0923] [0.0923] [0.2173]
[0.0923] [0.2173] [0.0923] [0.0923] [0.1903] [0.4147] [0.5292]
[0.0923] [0.0923] [0.0923] [0.0923] [0.0923] [0.0923] [0.0923]
[0.0923] [0.0923] [0.0923] [0.5127] [0.0923] [0.0923] [0.0923]
[0.0923] [0.0923] [0.1903] [0.0923] [0.4147] [0.0923] [0.0923]
[0.0923] [0.0923] [0.0923] [0.6542] [0.0923] [0.0923] [0.0923]

La matriz de visibilidad resultante es:
[0.0000] [0.0200] [0.0333] [0.0000] [0.0000] [0.0167] [0.0000]
[0.0200] [0.0000] [0.0250] [0.0000] [0.0000] [0.0000] [0.0200]
[0.0333] [0.0250] [0.0000] [0.0000] [0.0167] [0.0333] [0.0333]
[0.0000] [0.0000] [0.0000] [0.0000] [0.0200] [0.0000] [0.0250]
[0.0000] [0.0000] [0.0167] [0.0200] [0.0000] [0.0200] [0.0000]
[0.0167] [0.0000] [0.0333] [0.0000] [0.0200] [0.0000] [0.0000]
[0.0000] [0.0200] [0.0333] [0.0250] [0.0000] [0.0000] [0.0000]

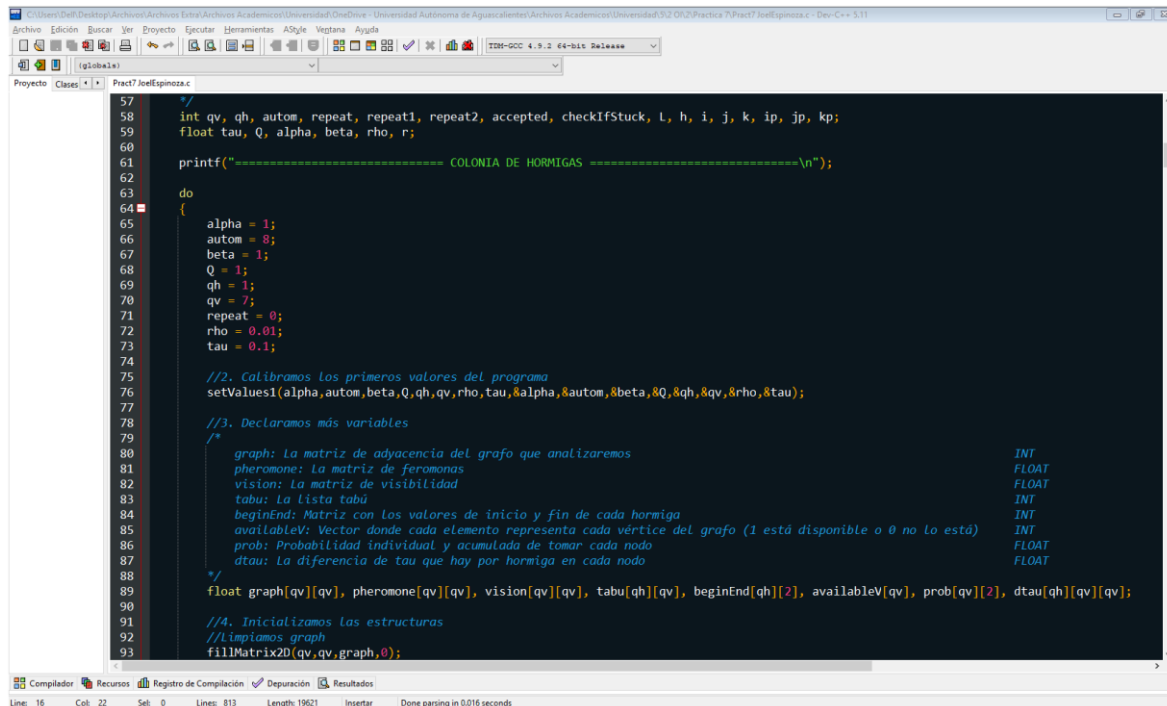
¿Desea repetir el código?
0. No
1. Sí

```

## Anexo 3: Algunas capturas del código en el IDE



```
21 #include <stdio.h>
22 #include <stdlib.h>
23 #include <locale.h>
24 #include <time.h>
25 #include <math.h>
26
27 void setValues1(float alpha, int autom, float beta, float Q, int qh, int qv, float rho, float tau, float *alphaP, int *automP, float *betaP, float *Q, int *qhP, int *qvP, float *rhoP, float *tauP, float *alphaPP, int *automPP, float *betaPP, float *QPP, int *qhPP, int *qvPP, float *rhoPP, float *tauPP);
28 void fillMatrix1D(int a, float matrix[a], float n);
29 void fillMatrix2D(int a, int b, float matrix[a][b], float n);
30 void fillMatrix3D(int a, int b, int c, float matrix[a][b][c], float n);
31 void setValues2(int qv, int qh, float graph[qv][qv], float pheromone[qv][qv], float vision[qv][qv], float beginEnd[qh][2]);
32 void startProcess(int i, int qh, int qv, float tabu[qh][qv], float beginEnd[qh][2], float availableV[qv]);
33 int acceptVertex(int i, int qh, int qv, int value, float tabu[qh][qv]);
34 int getL(int i, int qh, int qv, float graph[qv][qv], float tabu[qh][qv], float beginEnd[qh][2]);
35
36 main()
37 {
38     setlocale(LC_ALL, "");
39     srand(time(NULL));
40
41     //1. Declaramos las variables que usaremos
42     /*
43      qv: Cantidad de vértices del grafo
44      qh: Cantidad de hormigas
45      tau: Valor de la feromona inicial
46      Q: Parámetro
47      alpha: Parámetro
48      beta: Parámetro
49      rho: Parámetro
50      autom: Modo de acción (Para autom >= 1 se tomará el número de autom como las repeticiones)
51      repeat: Sirve para repetir todo el código nuevamente
52      repeat1: Sirve para repetir la colonia en modo manual
53      r: Número aleatorio
54      accepted: Bandera de aceptación de vértice
55      checkIfStuck: Iterador que no tolerará un número determinado de iteraciones. Si se alcanzan, se interpretará que la hormiga se encerró
56      L: Longitud del recorrido completo realizado por la hormiga
57
```



```
57
58 int qv, qh, autom, repeat, repeat1, repeat2, accepted, checkIfStuck, L, h, i, j, k, ip, jp, kp;
59 float tau, Q, alpha, beta, rho, r;
60
61 printf("===== COLONIA DE HORMIGAS =====\n");
62
63 do
64 {
65     alpha = 1;
66     autom = 0;
67     beta = 1;
68     Q = 1;
69     qh = 1;
70     qv = 7;
71     repeat = 0;
72     rho = 0.01;
73     tau = 0.1;
74
75     //2. Calibramos los primeros valores del programa
76     setValues1(alpha, autom, beta, Q, qh, qv, rho, tau, &alpha, &autom, &beta, &Q, &qh, &qv, &rho, &tau);
77
78     //3. Declaramos más variables
79     /*
80      graph: La matriz de adyacencia del grafo que analizaremos          INT
81      pheromone: La matriz de feromonas                                FLOAT
82      vision: La matriz de visibilidad                                FLOAT
83      tabu: La lista tabu                                              INT
84      beginEnd: Matriz con los valores de inicio y fin de cada hormiga    INT
85      availableV: Vector donde cada elemento representa cada vértice del grafo (1 está disponible o 0 no lo está)    INT
86      prob: Probabilidad individual y acumulada de tomar cada nodo        FLOAT
87      dtau: La diferencia de tau que hay por hormiga en cada nodo          FLOAT
88     */
89     float graph[qv][qv], pheromone[qv][qv], vision[qv][qv], tabu[qh][qv], beginEnd[qh][2], availableV[qv], prob[qv][2], dtau[qh][qv][qv];
90
91     //4. Inicializamos las estructuras
92     //limpiamos graph
93     fillMatrix2D(qv, qv, graph, 0);
```

```
90
91 //4. Inicializamos las estructuras
92 //limpiamos graph
93 fillMatrix2D(qv,qv,graph,0);
94
95 //Inicializamos pheromone
96 fillMatrix2D(qv,qv,pheromone,tau);
97
98 //limpiamos vision
99 fillMatrix2D(qv,qv,vision,0);
100
101 //limpiamos tabu
102 fillMatrix2D(qh,qv,tabu,0);
103
104 //limpiamos beginEnd
105 fillMatrix2D(qh,2,beginEnd,0);
106
107 //Inicializamos availableV
108 fillMatrix1D(qv,availableV,1);
109
110 //limpiamos prob
111 fillMatrix2D(qv,2,prob,0);
112
113 //5. Calibramos los demás valores del programa
114 setValues2(qv,qh,graph,pheromone,vision,beginEnd);
115
116 //Inicializamos vision
117 for(i = 0; i < qv; i++)
118 {
119     for(j = 0; j < qv; j++)
120     {
121         if(graph[i][j] == 0)
122         {
123             vision[i][j] = 0;
124         }
125         else
126         {
```

```
171
172 //6. Comenzamos la colonia de hormigas, la cual se repetirá tantas veces como se haya calibrado autom (si es 0, se decide el paro)
173 h = 0;
174 repeat2 = 1;
175 do
176 {
177     //Inicializamos dtau
178     fillMatrix3D(qh,qv,qv,dtau,0);
179
180     if(autom == 0)
181     {
182         printf("\n");
183         printf("\n");
184         printf("Comenzamos la %dª iteración\n\n", h + 1);
185     }
186
187     //El procedimiento de la colonia se hará para cada hormiga
188     i = 0;
189     do
190     {
191         if(autom == 0)
192         {
193             printf(" Comenzamos a trabajar con la hormiga %d\n", i + 1);
194             getchar();
195         }
196
197         //Inicializamos tabu y availableV con los valores proporcionados por beginEnd
198         startProcess(i,qh,qv,tabu,beginEnd,availableV);
199
200         if(autom == 0)
201         {
202             printf(" Inicializamos la lista tabu con el valor del nodo de inicio:\n ");
203             for(ip = 0; ip < qh; ip++)
204             {
205                 for(jp = 0; jp < qv; jp++)
206                 {
207                     printf("%d ", (int)tabu[ip][jp]);
```

```
222 //Todavía, el procedimiento de selección de camino de cada hormiga tiene que hacerse por cada nodo como máximo qv - 1 veces
223 j = 0;
224 do
225 {
226     if(autom == 0)
227     {
228         printf("\n");
229         printf("    Escogeremos el nodo %d\n", j + 1);
230     }
231
232     //Debemos repetir este análisis hasta que el nodo haya sido reconocido como válido
233     accepted = 0;
234     checkIfStuck = 0;
235     do
236     {
237         //A partir del nodo otorgado, vamos al renglón en vision con el que rellenaremos prob y procedemos a rellenar prob
238         prob[0][0] = vision[(int)tabu[i][j] - 1][0];
239         prob[0][1] = prob[0][0];
240         for(k = 1; k < qv; k++)
241         {
242             prob[k][0] = vision[(int)tabu[i][j] - 1][k];
243             prob[k][1] = prob[k][0] + prob[k - 1][1];
244         }
245
246         //Generamos un número aleatorio y discretizamos el valor entre 10000 posibilidades, todas dentro del rango
247         r = rand() % 10000;
248         r = r/(10000/prob[qv - 1][1]);
249
250         if(autom == 0)
251         {
252             printf("    Tenemos las probabilidades individuales y acumuladas de que de %d se vaya a\n", (int)tabu[i][j]);
253             for(ip = 0; ip < qv; ip++)
254             {
255                 printf("        %d:    ", ip + 1);
256                 for(jp = 0; jp < 2; jp++)
257                 {
258                     printf("[%4f]    ", prob[ip][jp]);
259                 }
260             }
261         }
262     } while(!accepted && !checkIfStuck);
263 }
```

```
264 //Ubicamos el nodo seleccionado
265 for(k = 0; k < qv; k++)
266 {
267     if(r < prob[k][1])
268     {
269         //Encontramos que cayó en k + 1
270         break;
271     }
272 }
273
274 if(autom == 0)
275 {
276     printf("    Hemos escogido el nodo %d (r = %4f)\n", k + 1, r);
277 }
278
279 //Ahora toca validar si ese men está disponible o no
280 accepted = acceptVertex(i, qh, qv, k + 1, tabu);
281
282 if(autom == 0)
283 {
284     if(accepted == 1)
285     {
286         printf("    Revisando el nodo %d con tabú, obtuvimos que éste es aceptado\n", k + 1);
287     }
288     else
289     {
290         printf("    Revisando el nodo %d con tabú, obtuvimos que éste no es aceptado\n", k + 1);
291     }
292 }
293
294 //También tenemos que validar si la hormiga no se quedó encerrada
295 if(accepted == 0)
296 {
297     checkIfStuck++;
298 }
299
300 if(autom == 0)
```

```
294
295 //También tenemos que validar si la hormiga no se quedó encerrada
296 if(accepted == 0)
297 {
298     checkIfStuck++;
299     if(autom == 0)
300     {
301         printf("    Tendremos que repetir la selección aleatoria por posible encrucijada (La hormiga lleva %d repeticio\n", j);
302     }
303 }
304
305 if(checkIfStuck == 100)
306 {
307     if(autom == 0)
308     {
309         printf("    Hemos interpretado que la hormiga se quedó atorada\n Reiniciaremos el proceso completo\n");
310     }
311     startProcess(i,qh,qv,tabu,beginEnd,availableV);
312     checkIfStuck = 0;
313     j = 0;
314 }
315 while(accepted == 0);
316
317 //Hemos aceptado el nodo, vamos a actualizar la lista tabú y availableV
318 j++;
319 tabu[i][j] = k + 1;
320 availableV[(int)tabu[i][j] - 1] = 0;
321
322 if(autom == 0)
323 {
324     printf("    Hemos aceptado el nodo. Ahora tenemos el camino siguiente\n    ");
325     for(ip = 0; ip < qh; ip++)
326     {
327         for(jp = 0; jp < qv; jp++)
328         {
329             if((int)tabu[i][j] == (int)beginEnd[ip][jp])
330             {
331                 printf("    Camino válido: %d %d\n", ip, jp);
332             }
333         }
334     }
335 }
```

```
342
343 //Chequemos si el nodo aceptado es el definido como el último. Así la hormiga se sentirá realizada
344 if(k + 1 == beginEnd[i][1])
345 {
346     if(autom == 0)
347     {
348         printf("    Hemos llegado al nodo objetivo\n");
349     }
350     break;
351 }
352 while(j < qv - 1);
353
354 L = getL(i,qh,qv,graph,tabu,beginEnd);
355
356 if(autom == 0)
357 {
358     printf("\n");
359     printf("    La hormiga produjo un recorrido de L = %d\n",L);
360     getchar();
361 }
362
363 if(autom != 0)
364 {
365     printf("En la iteración %d, la hormiga %d produjo el camino:\n",h + 1, i + 1);
366     for(jp = 0; jp < qv; jp++)
367     {
368         printf("%d] ", (int)tabu[ip][jp]);
369     }
370     printf("\n");
371     printf("Con un recorrido de valor %d\n",L);
372 }
373
374 for(j = 0; j < qv; j++)
375 {
376     if((int)tabu[i][j] == (int)beginEnd[i][1])
377     {
378         printf("    Camino válido: %d %d\n", i, j);
379     }
380 }
```

```
387     if(autom == 0)
388     {
389         printf("    Se ha generado el siguiente incremento de feromonas:\n    ");
390         for(jp = 0; jp < qv; jp++)
391         {
392             for(kp = 0; kp < qv; kp++)
393             {
394                 printf("[%4f] ", dtau[i][jp][kp]);
395             }
396             printf("\n    ");
397         }
398         getchar();
399     }
400
401     i++;
402 }
403 while(i < qh);
404
405 //Las hormigas han llegado al nodo destino, ahora procedemos a actualizar pheromone
406 for(j = 0; j < qv; j++)
407 {
408     for(k = 0; k < qv; k++)
409     {
410         for(i = 0; i < qh; i++)
411         {
412             dtau[0][j][k] = dtau[0][j][k] + dtau[i][j][k];
413             pheromone[j][k] = ((1-rho)*(pheromone[j][k])) + dtau[0][j][k];
414         }
415     }
416 }
417
418 if(autom == 0)
419 {
420     printf("    Tras el incremento y evaporación de feromonas, la nueva matriz de feromonas es:\n    ");
421     for(jp = 0; jp < qv; jp++)
422     {
423         for(kp = 0; kp < qv; kp++)
```

```
432
433
434 //La colonia de hormigas ha finalizado, preguntamos por una nueva iteración o terminar el programa, según sea el caso
435 if(autom == 0)
436 {
437     printf("¿Desea usar un nuevo grupo de hormigas?\n");
438     printf("0. No\n");
439     printf("1. Si\n");
440     scanf("%d", &repeat1);
441     if(repeat1 == 0)
442     {
443         repeat2 = 0;
444     }
445     if(repeat1 == 1)
446     {
447         repeat2 = 1;
448     }
449     h++;
450 }
451 else
452 {
453     h++;
454     if(h == autom)
455     {
456         repeat2 = 0;
457     }
458 }
459 }
460 while(repeat2 == 1);
461
462 printf("===== Reporte Final =====\n");
463 printf("Hemos analizado el grafo:\n");
464 for(ip = 0; ip < qv; ip++)
465 {
466     for(jp = 0; jp < qv; jp++)
467     {
468         printf("[%d] ", (int)graph[ip][jp]);
```