Tecnológico de Monterrey

“Actividad Integral de grafos”

Programación de estructuras de datos y algoritmos fundamentales

Profesora: María Valentina Narváez Terán

Bruno Fernando Zabala Peña - A00838627

24/11/2024

Problemática

En un mundo que está en constante movimiento, la industria del food delivery está tomando un impulso bastante importante, pues ofrece ventajas tanto para clientes y proveedores por igual, tales como:

- Abarcar un mercado mucho más amplio sin la necesidad de una expansión física de un negocio.
- Tener acceso a alimentos sin la necesidad de transportarse.
- Ahorro de tiempo al no tener la necesidad de cocinar.
- Mejorar la imagen de un negocio y hacerlo mucho más competitivo.
- Aumentar las ventas.
- Etc.

Sin embargo, y aunque existen herramientas y plataformas que están haciendo cada vez más sencillo adentrarse en este modelo de ventas (Uber Eats, Rappi, etc.), los negocios aún deben contar con una infraestructura o sistemas que le permitan llevar un monitoreo de lo que implica este servicio, como el registro de ventas o clientes.

En esta primera etapa de la solución, se creó una aplicación capaz de manejar un conjunto de órdenes de food delivery, ordenarlas de manera cronológica y permitir al usuario la búsqueda de datos específicos.

Código

Para esta etapa, el programa debía cumplir con las siguientes funciones a través del uso de grafos:

- Búsquedas de menú por restaurantes y por platos.
- Cálculo de la ruta más eficiente para una entrega.

La ventaja de trabajar con este tipo de estructura de datos es la versatilidad que ofrecen para representar distintas situaciones o escenarios que implican algún tipo de conexión. En este caso, se utilizaron grafos dirigidos y ponderados para representar el menú de cada uno de los restaurantes, además de grafos

ponderados no dirigidos para representar una ciudad de 30x30 nodos y la distancia entre cada uno de estos.

Antes de desarrollar el funcionamiento del programa, se leen los archivos de texto que contienen la información necesaria para crear los grafos y realizar las pruebas.

```
// Archivos de entrada
ifstream menus("menus.txt");
ifstream city30x30("city30x30.txt");
ifstream restaPlaces("restaPlaces.txt");
ifstream orders("orders-city30x30.txt");
```

Para generar el grafo de restaurantes y platillos, se creó una matriz de adyacencia a partir de las líneas obtenidas del archivo "menus.txt". La primera línea contiene el número de restaurantes y de platillos que están disponibles, los cuales, al sumarse, dan como resultado el número de nodos existentes en el grafo. Con esta información, es posible inicializar la matriz con las dimensiones correspondientes.

```
if(menus.is_open()){
    // Lee cada linea del archivo de menus
    while(getline(menus, rest)){
        if(i == 0){
            // cout << "Informacion: " << rest << endl;
            // Obtiene el numero de restaurantes y platos
            int index = rest.find(" ");
            aux = rest.substr(0, index);
            num_restaurants = stoi(aux);

            rest = rest.substr(index + 1);
            index = rest.find(" ");
            aux = rest.substr(0, index);
            num_dishes = stoi(aux);

            // Crea la matriz de adyacencia
            N = num_restaurants + num_dishes;

            menus_matrix = (int **) calloc (N, sizeof(int*));

            for(j = 0; j < N; j++){
                menus_matrix[j] = (int *) calloc(N, sizeof(int));
            }

            i++;
        }
```

Al cambiar de línea, se comienzan a leer cada una de las entradas, de las cuales se extraen el restaurante (nodo 1), platillo (nodo 2) y precio (peso) con las funciones get_restaurant, get_dish y get_revenue, respectivamente. Estos primeros dos son

almacenados en una lista enlazada doble para que sus índices, guardados en a y b, puedan ser utilizados en la matriz de adyacencia. Por último, la matriz guarda en la posición [a][b] el precio del platillo para indicar que este existe en el menú del restaurante.

```
// Funcion para obtener el nombre del restaurante de una orden
// Complejidad: O(1)
string get_restaurant(string order){
    string restaurant;

    // Encuentra los indices en los que inicia y termina el nombre del restaurante
    int restaurant_start_index = order.find("R:");
    int restaurant_end_index = order.find(" O:");

    // Extrae el nombre del restaurante
    restaurant = order.substr(restaurant_start_index + 2, restaurant_end_index - restaurant_start_index - 2);

    return restaurant;
}
```

```
// Funcion para obtener el nombre del plato de una orden
// Complejidad: O(1)
string get_dish(string order){
    string dish;

    // Encuentra los indices en los que inicia y termina el nombre del plato
    int dish_start_index = order.find("O:");
    int dish_end_index = order.find("(");

    // Extrae el nombre del plato
    dish = order.substr(dish_start_index + 2, dish_end_index - dish_start_index - 2);

    return dish;
}
```

```
// Funcion para obtener la ganancia de una orden
// Complejidad: O(1)
float get_revenue(string order){
    int revenue_start_index = order.find("(");
    int revenue_end_index = order.find(")");

    string revenue = order.substr(revenue_start_index + 1, revenue_end_index - revenue_start_index - 1);

    return stof(revenue);
}
```

```
else{
    // Obtiene los restaurantes y sus platos
    string restaurant = get_restaurant(rest);
    string dish = get_dish(rest);
    float revenue = get_revenue(rest);

    // Agrega el restaurante y el plato a la lista de datos si no estan ya en ella
    if(datos.find(restaurant) == -1){
        datos.insertLast(restaurant);
    }

    if(datos.find(dish) == -1){
        datos.insertLast(dish);
    }

    a = datos.find(restaurant);
    b = datos.find(dish);

    // cout << "a: " << a << " b: " << b << endl;

    // Agrega la relacion entre restaurantes y platos a la matriz de adyacencia
    // Agrega el precio del plato
    menus_matrix[a][b] = revenue;
}
```

Una vez que se tiene el grafo generado, se le permite al usuario elegir la opción de buscar el menú de un restaurante específico o verificar la existencia de un platillo en cada uno de los restaurantes. Dependiendo de la opción elegida, se busca en la lista de datos el nombre del restaurante o platillo solicitado por el usuario, el índice se utiliza para verificar la columna o fila del mismo y se muestran aquellos espacios (relaciones) en los que haya un valor diferente a 0, lo cual indica la existencia de un platillo en un restaurante.

```
switch(option){
    case 1:
        cout << "Menu de un restaurante en especifico" << endl;
        cout << "Seleccione un restaurante: ";
        getline(cin >> ws, restaurant);
        cout << endl;

        // Obtiene el indice del restaurante en la lista de datos
        restaurant_index = datos.find(restaurant);

        cout << "--------------------------------" << endl;
        cout << "Menu de " << restaurant << endl;
        cout << "--------------------------------" << endl;
        cout << endl;

        // Muestra los platos que ofrece el restaurante (espacios donde hay un precio)
        for(i = 0; i < N; i++){
            if(menus_matrix[restaurant_index][i] != 0){
                cout << "Plato: " << datos.get(i) << " (Precio: " << menus_matrix[restaurant_index][i] << ")" << endl;
            }
        }

        break;
```

```
case 2:
    cout << "Restaurantes que ofrecen un plato en especifico" << endl;
    cout << "Seleccione un plato: ";
    getline(cin >> ws, dish);
    cout << endl;

    // Obtiene el indice del plato en la lista de datos
    dish_index = datos.find(dish);

    cout << "-----------------------------------------------" << endl;
    cout << "Restaurantes que ofrecen " << dish << endl;
    cout << "-----------------------------------------------" << endl;
    cout << endl;

    // Muestra los restaurantes que ofrecen el plato (espacios donde haya un 1)
    for(i = 0; i < N; i++){
        if(menus_matrix[i][dish_index] != 0){
            cout << "Restaurante: " << datos.get(i) << " (Precio: " << menus_matrix[i][dish_index] << ")" << endl;
        }
    }

    break;
```

Para las rutas de entrega, primero se generó una matriz de adyacencia a partir del contenido del archivo “city30x30.txt”. Para crearla, se siguió un proceso bastante similar al de la matriz de los restaurantes y platillos, con la diferencia de que

1. Ya se conocía la cantidad de nodos desde un inicio (900 al ser una ciudad 30x30).
2. El peso entre nodos también se guarda en la posición [b][a] al ser un grafo no dirigido.

```
// Lista de posiciones de la ciudad
List<string> city_positions;

// Variables para la matriz de adyacencia de la ciudad
string coord;
int **city_matrix;
N = 900; // max: (29, 29)

// Matriz de adyacencia para representar la ciudad
if(city30x30.is_open()){
    // Crea la matriz de adyacencia
    city_matrix = (int **) calloc (N, sizeof(int*));

    for(j = 0; j < N; j++){
        city_matrix[j] = (int *) calloc(N, sizeof(int));
    }
```

```
// Lee cada linea del archivo de la ciudad
// Formato de linea (0, 0) (1, 0) 50
while(getline(city30x30, coord)){
    // Obtiene los nodos y el peso
    int index = coord.find(")");
    aux = coord.substr(0, index + 1);
    // cout << "aux: " << aux << endl;

    coord = coord.substr(index + 1);
    index = coord.find(")");
    string aux2 = coord.substr(1, index);
    // cout << "aux2: " << aux2 << endl;

    coord = coord.substr(index + 2);
    int w = stoi(coord);
    // cout << "w: " << w << endl;

    // Agrega los nodos a la lista de posiciones si no estan ya en ella
    if(city_positions.find(aux) == -1){
        city_positions.insertLast(aux);
    }

    if(city_positions.find(aux2) == -1){
        city_positions.insertLast(aux2);
    }

    a = city_positions.find(aux);
    b = city_positions.find(aux2);

    // Agrega el peso entre los nodos a la matriz
    city_matrix[a][b] = w;
    // Grafo no dirigido
    city_matrix[b][a] = w;
}
```

Ya con el grafo generado, se puede pasar a la resolución de los casos de prueba. Para esto, se leyó el contenido del archivo “restaPlaces.txt” y los nombres de los restaurantes y sus coordenadas se almacenaron en listas individuales.

```
cout << endl << "Rutas" << endl;

// Listas para almacenar los restaurantes y sus posiciones
List<string> restaurants;
List<string> restaurant_positions;
int index;

// Lee cada linea del archivo de los restaurantes
while(getline(restaPlaces, rest)){
    index = rest.find(" (");
    restaurant = rest.substr(0, index);

    // Agrega los restaurantes y sus posiciones a las listas
    restaurants.insertLast(restaurant);
    restaurant_positions.insertLast(get_coordinates(rest));
}

restaPlaces.close();
```

Después, se leen los casos de prueba desde el archivo “orders-city30x30.txt”. De cada línea se extrae y muestra en la consola:

1. El platillo.
2. La fecha de la orden.
3. El nombre del restaurante.
4. Las coordenadas de inicio (posición del restaurante).
5. Las coordenadas para la entrega (posición final).

```
// Variables para los casos de prueba
i = 1;
string position;
string end_position;
string date;

// Lee cada linea del archivo de las ordenes
while(getline(orders, rest)){
    // Extrae la informacion de la orden
    restaurant = get_restaurant(rest);
    end_position = get_coordinates(rest);
    dish = get_dish(rest);
    date = get_date(rest);

    // Obtiene el indice del restaurante en la lista de restaurantes
    index = restaurants.find(restaurant);

    // Obtiene la posicion del restaurante
    position = restaurant_positions.get(index);

    cout << "Caso de prueba " << i << endl;
    cout << "----------------" << endl;
    cout << "Orden: " << dish << endl;
    cout << "Fecha: " << date << endl;
    cout << "Restaurante de entrega: " << restaurant << " " << position << endl;
    cout << "Lugar de entrega: " << end_position << endl;

    // Obtiene el camino mas corto entre el restaurante y el lugar de entrega
    dijkstra(city_matrix, city_positions, position, end_position, N);

    cout << endl;

    i++;
}
```

Por último, para calcular la ruta de entrega más óptima, se implementó el algoritmo de Dijkstra, el cual funciona de la siguiente manera:

1. Se crean listas para guardar las distancias, los nodos procesados y los predecesores.
2. Las distancias se inicializan en infinito a excepción de la del origen y se marcan todos los nodos como no visitados.
3. Se selecciona el nodo con la menor distancia no procesado y se marca como visitado. Este nodo pasa a ser el nodo actual.
4. Para cada nodo vecino del nodo actual, se calcula la distancia desde el nodo origen pasando por el nodo actual. Si esta distancia es menor que la distancia almacenada previamente para ese vecino, se actualiza la distancia y se guarda el nodo actual como predecesor.

5. El proceso se repite seleccionando el siguiente nodo con la menor distancia no procesado, hasta que todos los nodos hayan sido visitados o el nodo destino tenga la distancia mínima calculada.
6. Se despliega la distancia mínima encontrada entre el nodo inicial y el final, así como el peso total, su recorrido y la cantidad de aristas que componen al mismo.

```
// Algoritmo de Dijkstra para encontrar el camino mas corto entre dos nodos
// Complejidad O((V + E) log V)
void dijkstra(int **matrix, List<string> posiciones, string start, string end, int N){
    // Arreglo para guardar la distancia mas corta
    int *dist = (int *) calloc(N, sizeof(int));

    // Arreglo para guardar si el nodo ya fue visitado
    bool *visited = (bool *) calloc(N, sizeof(bool));

    // Arreglo para guardar predescesores
    int *pred = (int *) calloc(N, sizeof(int));

    // Inicia las distancias en infinito y los nodos como no visitados
    for(int i = 0; i < N; i++){
        dist[i] = INT_MAX;
        visited[i] = false;
    }

    // Indices de las posiciones de inicio y fin
    int start_index = posiciones.find(start);
    int end_index = posiciones.find(end);

    // La distancia del nodo de inicio a si mismo es 0
    dist[start_index] = 0;
```

```
// Encuentra el camino mas corto
for(int i = 0; i < N - 1; i++){
    int min = INT_MAX;
    int min_index;

    for(int j = 0; j < N; j++){
        if(visited[j] == false && dist[j] <= min){
            min = dist[j];
            min_index = j;
        }
    }

    // Marca el nodo como visitado
    visited[min_index] = true;

    // Actualiza la distancia de los nodos adyacentes
    for(int j = 0; j < N; j++){
        if(!visited[j] && matrix[min_index][j] && dist[min_index] != INT_MAX && dist[min_index] + matrix[min_index][j] < dist[j]){
            dist[j] = dist[min_index] + matrix[min_index][j];
            pred[j] = min_index;
        }
    }
}
```

```cpp
// Muestra la distancia mas corta
cout << "La distancia mas corta entre " << posiciones.get(start_index) << " y " << posiciones.get(end_index) << " es: " << dist[end_index] << endl;

// Muestra el camino mas corto
cout << "Camino: " << posiciones.get(end_index) << " ";
int j = end_index;
int num_aristas = 0;
while(j != start_index){
    cout << posiciones.get(pred[j]) << " ";
    j = pred[j];
    num_aristas++;
}

cout << endl << "Numero de aristas: " << num_aristas << endl;
```

Pruebas

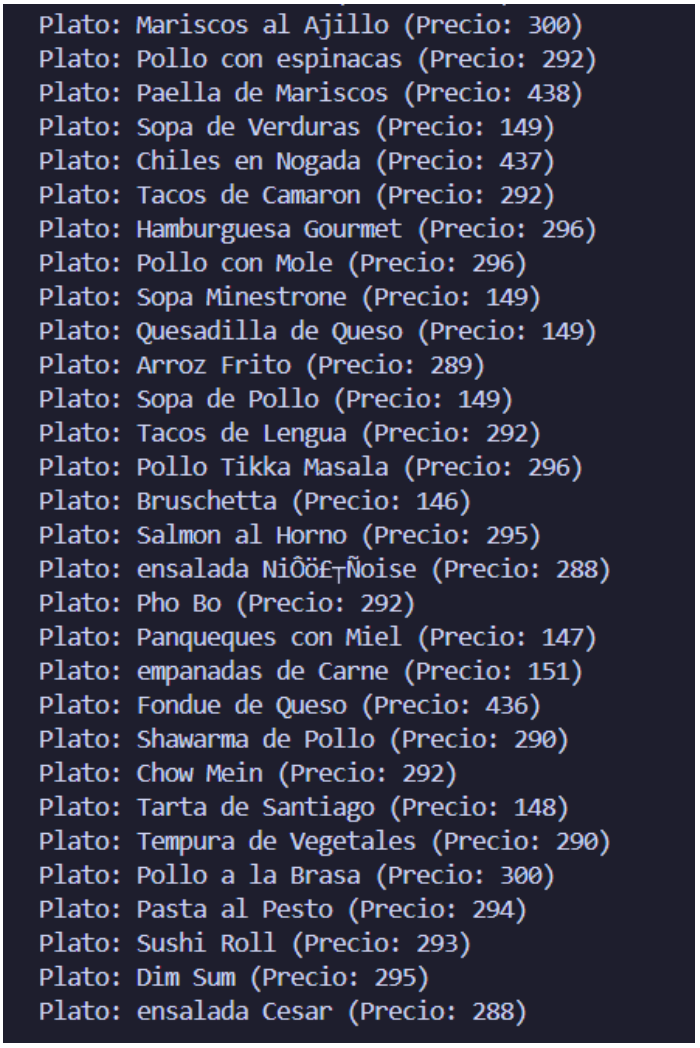
Búsqueda del menú de un restaurante

```
Menus
1- Busqueda del menu de un restaurante en especifico
2- Busqueda de restaurantes que ofrecen un plato en especifico
> 1

Menu de un restaurante en especifico
Seleccione un restaurante: La Parrilla Argentina

--------------------------------
Menu de La Parrilla Argentina
--------------------------------

Plato: ensalada Griega (Precio: 150)
Plato: Curry de Pollo (Precio: 296)
Plato: Fettuccine Alfredo (Precio: 295)
Plato: ensalada Caprese (Precio: 150)
Plato: Mole Poblano (Precio: 295)
Plato: Goulash Hungaro (Precio: 436)
Plato: Tortellini Alfredo (Precio: 293)
Plato: Sopa de Pescado (Precio: 436)
Plato: Cochinita Pibil (Precio: 295)
Plato: Cazuela de Mariscos (Precio: 440)
Plato: Chop Suey (Precio: 293)
Plato: Pizza Margherita (Precio: 291)
Plato: Sopa de Tortilla (Precio: 148)
Plato: Ratatouille (Precio: 293)
Plato: Gambas al Ajillo (Precio: 303)
Plato: Osso Buco (Precio: 445)
Plato: Pasta Primavera (Precio: 293)
Plato: Ensalada de Pollo (Precio: 289)
Plato: Carne de Res en Salsa (Precio: 300)
Plato: Fajitas de Res (Precio: 298)
Plato: Lasana Vegetariana (Precio: 296)
Plato: Tacos de Birria (Precio: 292)
Plato: Risotto de Champinones (Precio: 436)
Plato: Cordero Asado (Precio: 444)
```



Búsqueda de la existencia de un platillo en los restaurantes

```
Menus
1- Busqueda del menu de un restaurante en especifico
2- Busqueda de restaurantes que ofrecen un plato en especifico
> 2

Restaurantes que ofrecen un plato en especifico
Seleccione un plato: Fettuccine Alfredo

---------------------------------------------
Restaurantes que ofrecen Fettuccine Alfredo
---------------------------------------------

Restaurante: La Terraza del Mar (Precio: 288)
Restaurante: El Barzon (Precio: 276)
Restaurante: El Pueblo (Precio: 278)
Restaurante: La Abuela (Precio: 282)
Restaurante: Cocina Miguel (Precio: 278)
Restaurante: La Brasserie Parisienne (Precio: 267)
Restaurante: The Trendy eatery (Precio: 293)
Restaurante: La Terraza (Precio: 258)
Restaurante: La Terraza del Chef (Precio: 301)
Restaurante: El Rinconcito Tropical (Precio: 299)
Restaurante: El Grande (Precio: 293)
Restaurante: La Taberna del Chef (Precio: 282)
Restaurante: The Main (Precio: 299)
Restaurante: The Trendy Cafe (Precio: 284)
Restaurante: La Parrilla Argentina (Precio: 295)
Restaurante: The American (Precio: 271)
Restaurante: The Garden Table (Precio: 284)
Restaurante: The Mediterranean Table (Precio: 265)
Restaurante: Ocean (Precio: 256)
Restaurante: La Francesa (Precio: 261)
Restaurante: La Cantina de Juan (Precio: 284)
Restaurante: The Not Italian Bistro (Precio: 312)
```

```
Restaurante: The Rustic (Precio: 301)
Restaurante: The Taste (Precio: 261)
Restaurante: The Food Haven (Precio: 265)
Restaurante: The Modern Palate (Precio: 273)
Restaurante: El Restaurante de la esquina (Precio: 293)
Restaurante: Los Rincones (Precio: 252)
Restaurante: La esquina del Chef (Precio: 256)
Restaurante: The elegant Bistro (Precio: 306)
Restaurante: El Cafe de la Abuela (Precio: 261)
Restaurante: El Restaurante de la Plaza (Precio: 280)
Restaurante: The Rustic Table (Precio: 258)
Restaurante: El Rincon del Mar (Precio: 261)
Restaurante: The Caffeine Spot (Precio: 250)
Restaurante: The Artisan Table (Precio: 254)
Restaurante: La Rosticeria (Precio: 278)
Restaurante: Rios (Precio: 295)
Restaurante: La Cocina de Mama (Precio: 261)
Restaurante: Bosque Real (Precio: 282)
Restaurante: The eatery (Precio: 273)
Restaurante: El dragon (Precio: 267)
Restaurante: La Cocina del Chef (Precio: 261)
Restaurante: The Bistro Lounge (Precio: 293)
Restaurante: The Savory (Precio: 276)
```

Rutas de entrega

```
Rutas
Caso de prueba 1
----------------
Orden: Falafel con Hummus
Fecha: Jun 24 19:39:45
Restaurante de entrega: La Terraza del Mar (14, 27)
Lugar de entrega: (20, 29)
La distancia mas corta entre (14, 27) y (20, 29) es: 520
Camino: (20, 29) (19, 29) (18, 29) (18, 28) (18, 27) (17, 27) (16, 27) (15, 27) (14, 27)
Numero de aristas: 8
```

```
Caso de prueba 2
----------------
Orden: ensalada Griega
Fecha: Feb 13 19:25:24
Restaurante de entrega: El Barzon (12, 25)
Lugar de entrega: (19, 10)
La distancia mas corta entre (12, 25) y (19, 10) es: 1330
Camino: (19, 10) (18, 10) (18, 11) (18, 12) (18, 13) (18, 14) (18, 15) (18, 16) (17, 16) (17, 17) (17, 18) (16, 18) (16, 19) (15, 19) (14, 19) (14, 20) (13, 20)
 (13, 21) (13, 22) (13, 23) (13, 24) (13, 25) (12, 25)
Numero de aristas: 22
```

```
Caso de prueba 3
----------------
Orden: Curry de Pollo
Fecha: Feb 7 17:8:30
Restaurante de entrega: El Cafe de la Plaza (4, 4)
Lugar de entrega: (7, 2)
La distancia mas corta entre (4, 4) y (7, 2) es: 280
Camino: (7, 2) (7, 3) (6, 3) (6, 4) (5, 4) (4, 4)
Numero de aristas: 5
```

```
Caso de prueba 4
---------------
Orden: Tiradito de Pescado
Fecha: Dic 24 14:50:39
Restaurante de entrega: The Rustic Spoon (27, 19)
Lugar de entrega: (19, 10)
La distancia mas corta entre (27, 19) y (19, 10) es: 1010
Camino: (19, 10) (20, 10) (21, 10) (22, 10) (22, 11) (22, 12) (22, 13) (22, 14) (22, 15) (22, 16) (22, 17) (23, 17) (24, 17) (25, 17) (26, 17) (27, 17) (27, 18)
 (27, 19)
Numero de aristas: 17
```

```
Caso de prueba 5
---------------
Orden: Lasana Bolonesa
Fecha: Abr 27 18:6:57
Restaurante de entrega: La esquina del Taco (23, 18)
Lugar de entrega: (0, 15)
La distancia mas corta entre (23, 18) y (0, 15) es: 1600
Camino: (0, 15) (1, 15) (2, 15) (3, 15) (4, 15) (5, 15) (6, 15) (7, 15) (8, 15) (9, 15) (10, 15) (11, 15) (12, 15) (13, 15) (14, 15) (15, 15) (16, 15) (17, 15)
(18, 15) (19, 15) (20, 15) (21, 15) (21, 16) (22, 16) (22, 17) (23, 17) (23, 18)
Numero de aristas: 26
```

Referencias

Findus Foodservice & https://www.findusfoodservices.es/humans.txt. (s. f.). *¿Qué es el Delivery Food?* Findus Foodservices. https://www.findusfoodservices.es/novedades/que-es-el-delivery-food/

Anexos

Repositorio (carpeta “Etapa 4”): https://github.com/burnoz/situacion-problema