Tecnológico de Monterrey

“Actividad Integral de hashing”

Programación de estructuras de datos y algoritmos fundamentales

Profesora: María Valentina Narváez Terán

Bruno Fernando Zabala Peña - A00838627

28/11/2024

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 se implementó el uso de hashmaps para cumplir con los requisitos solicitados, los cuáles se mantuvieron bastante similares respecto a la entrega anterior:

- Búsquedas de menú por restaurantes y por platos.
- Búsqueda de restaurantes más cercanos a la posición del usuario.

La ventaja principal de utilizar hashmaps es la facilidad que tienen para almacenar y acceder a grandes cantidades de datos, lo cual resulta bastante conveniente para este proyecto, pues se está trabajando con un total de 163 restaurantes y 3638

platillos diferentes. Además, ofrecen una gran flexibilidad para representar distintas situaciones al poder guardar distintos valores en las llaves, en este caso, por ejemplo, se utilizó para representar menús, restaurantes donde se ofrece cierto plato y las posiciones donde se encuentran estos.

Antes de desarrollar el funcionamiento del programa, se leen los archivos de texto que contienen la información necesaria para crear los hashmaps y el grafo que representa a la ciudad.

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

Para los menús de los restaurantes y la lista de restaurantes donde se ofrece un plato, se generaron dos hashmap que guardan otro hashmap ([Key: [Key_2: Contenido]]). Para generar estos, se lee el contenido del archivo “menus.txt”. La primera línea contiene información sobre el número de platos y restaurantes, por lo que se ignora para la creación de estas estructuras. Después de saltar esta línea, se extrae toda la información necesaria de las siguientes con las funciones get_restaurant, get_dish y get_revenue. Se busca el restaurante en el hash correspondiente y si este aún no es una llave, se crea un nuevo hash con el platillo y su precio y estos se insertan al hash principal, en el caso contrario, el platillo y su precio se insertan directamente en el hash interno. Se sigue el mismo procedimiento para el hash de platillos.

```
// Hash que guarda un hash de platos y su precio para cada restaurante
Hash<Hash<int>*> restaurants(239);

// Hash que guarda un hash de restaurantes y su precio para cada plato
Hash<Hash<int>*> dishes(3911);
```

```
// Lee el archivo de menus
if(menus.is_open()){
    while(getline(menus, line)){
        if(k == 0){
            k++;
        }

        else{
            // Obtiene el restaurante, platillo y precio
            restaurant = get_restaurant(line);
            dish = get_dish(line);
            price = get_revenue(line);

            index = restaurants.find(restaurant);

            if(index == -1){
                // cout << "Insertando restaurante: " << restaurant << endl;

                Hash<int> *inner_hash = new Hash<int>(9973);
                inner_hash->insert(dish, price);
                restaurants.insert(restaurant, inner_hash);
            }

            else{
                // cout << "Insertando platillo: " << dish << " en " << restaurant << endl;
                restaurants.table[index]->item->insert(dish, price);
            }
```

```
index = restaurants.find(restaurant);

if(index == -1){
    // cout << "Insertando restaurante: " << restaurant << endl;

    Hash<int> *inner_hash = new Hash<int>(9973);
    inner_hash->insert(dish, price);
    restaurants.insert(restaurant, inner_hash);
}

else{
    // cout << "Insertando platillo: " << dish << " en " << restaurant << endl;
    restaurants.table[index]->item->insert(dish, price);
}

index = dishes.find(dish);

if(index == -1){
    // cout << "Insertando platillo: " << dish << endl;

    Hash<int> *inner_hash = new Hash<int>(3643);
    inner_hash->insert(restaurant, price);
    dishes.insert(dish, inner_hash);
}

else{
    // cout << "Insertando restaurante: " << restaurant << " en " << dish << endl;
    dishes.table[index]->item->insert(restaurant, price);
}
```

```
// 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);
}
```

Ya con estos generados, se le da la opción al usuario de

1. Ver el menú de un restaurante
2. Ver los restaurantes que ofrecen un plato
3. Continuar

Para las primeras dos opciones, se le solicita al usuario una llave del restaurante o platillo a buscar y esta se busca en el hash correspondiente. Si la encuentra, se llama a la función show para mostrar el contenido del hash interior en la posición del restaurante o plato solicitado.

```
cout << "Opciones" << endl;
cout << "1- Busqueda del menu de un restaurante en especifico" << endl;
cout << "2- Busqueda de restaurantes que ofrecen un plato en especifico" << endl;
cout << "3- Continuar" << endl;
cout << "> ";
int option;
cin >> option;
cout << endl;

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
        index = restaurants.find(restaurant);

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

        // Muestra los platos que ofrece el restaurante
        restaurants.table[index]->item->show();

        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
    index = dishes.find(dish);

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

    // Muestra los restaurantes que ofrecen el plato
    dishes.table[index]->item->show();

    break;
```

Si se elige la tercera opción, o después de mostrar el contenido solicitado, el programa continúa con la función de buscar restaurantes cercanos. Primero se generó una matriz de adyacencia a partir del contenido del archivo “city30x30.txt”. De cada línea se extraen los dos nodos (representados como coordenadas) y el peso que existe entre ellos. Esta información es almacenada en dos listas enlazadas para que sus índices puedan ser utilizados en la matriz de adyacencia.

Por último, se guarda en la matriz de adyacencia el peso entre los nodos en la posición [a][b] y [b][a].

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

// Variables para la matriz de adyacencia de la ciudad
string coord;
string aux;
int i = 0;
int **city_matrix;
int j;
int a;
int b;
int 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;
}
```

A partir del archivo “restaPlaces.txt”, se creó un hashmap que contiene las posiciones de los restaurantes.

```
// Hash de restaurantes y sus coordenadas
Hash<string> restaurant_coordinates(239);

// Lee el archivo de restaurantes y sus coordenadas
if(restaPlaces.is_open()){
    while(getline(restaPlaces, coord)){
        // Obtiene el nombre del restaurante y sus coordenadas
        index = coord.find(" (");
        restaurant = coord.substr(0, index);
        string coordinates = get_coordinates(coord);

        // Inserta el restaurante y sus coordenadas en el hash
        restaurant_coordinates.insert(restaurant, coordinates);
    }
}

restaPlaces.close();
```

Para la búsqueda de restaurantes cercanos, se le solicita al usuario una posición inicial en forma de coordenadas, la cuál luego se utiliza para la búsqueda por medio de Dijkstra.

```
// Busqueda de restaurantes cercanos
string start_position;
List<string> closest;

cout << "Busqueda de restaurantes cercanos" << endl;
cout << "Seleccione una posicion de inicio (formato: (0, 0)): ";
getline(cin >> ws, start_position);
cout << endl;
```

Para la búsqueda restaurantes, se utiliza 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.

Ya con las distancias calculadas, el programa busca las distancias a las posiciones de los restaurantes a partir de las coordenadas contenidas en el hash creado

anteriormente. De estas, se extraen las 3 más pequeñas y se devuelve al usuario la lista de restaurantes más cercanos.

```
// Algoritmo de Dijkstra para encontrar el camino mas corto entre dos nodos
// Complejidad O((V + E) log V)
List<string> dijkstra(int **matrix, List<string> posiciones, string start, Hash<string> res_positions, 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;
    }

    // Indice de las posicion de inicio
    int start_index = posiciones.find(start);

    // 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;
        }
    }
}
```

```
// Lista de restaurantes mas cercanos
List<string> closest;

string min1 = "";
string min2 = "";
string min3 = "";

int min1_index = -1;
int min2_index = -1;
int min3_index = -1;

// Encuentra los 3 restaurantes mas cercanos
for(int i = 0; i < res_positions.maxSize; i++){
    if(res_positions.table[i]->free == false){
        string coordinates = res_positions.table[i]->item;
        int index = posiciones.find(coordinates);

        if(dist[index] < dist[min1_index] || min1_index == -1){
            min3 = min2;
            min3_index = min2_index;

            min2 = min1;
            min2_index = min1_index;

            min1 = res_positions.table[i]->str_key;
            min1_index = index;
        }

        else if(dist[index] < dist[min2_index] || min2_index == -1){
            min3 = min2;
            min3_index = min2_index;

            min2 = res_positions.table[i]->str_key;
            min2_index = index;
        }

        else if(dist[index] < dist[min3_index] || min3_index == -1){
            min3 = res_positions.table[i]->str_key;
            min3_index = index;
        }
    }
}
```

```
cout << "Restaurantes mas cercanos a la posicion " << start << endl;
cout << "1. " << min1 << " a " << dist[min1_index] << " metros" << endl;
cout << "2. " << min2 << " a " << dist[min2_index] << " metros" << endl;
cout << "3. " << min3 << " a " << dist[min3_index] << " metros" << endl;

closest.insertLast(min1);
closest.insertLast(min2);
closest.insertLast(min3);

return closest;
```

Sabiendo los restaurantes más cercanos, se le da la opción al usuario de consultar el menú de alguno de estos, si se elige proceder, se solicita el restaurante que se quiere consultar y se despliega su menú utilizando el hash de restaurantes.

```
closest = dijkstra(city_matrix, city_positions, start_position, restaurant_coordinates, N);

cout << endl;
cout << "Consultar menu de los restaurantes" << endl;
cout << "1. Si" << endl;
cout << "2. No" << endl;
cout << "> ";
cin >> option;

switch(option){
    case 1:
        cout << "Seleccione un restaurante de los encontrados: ";
        cin >> option;
        cout << endl;

        // Obtiene el nombre del restaurante
        restaurant = closest.get(option - 1);

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

        // Muestra el menu del restaurante
        restaurants.table[restaurants.find(restaurant)]->item->show();

        break;
```

Pruebas

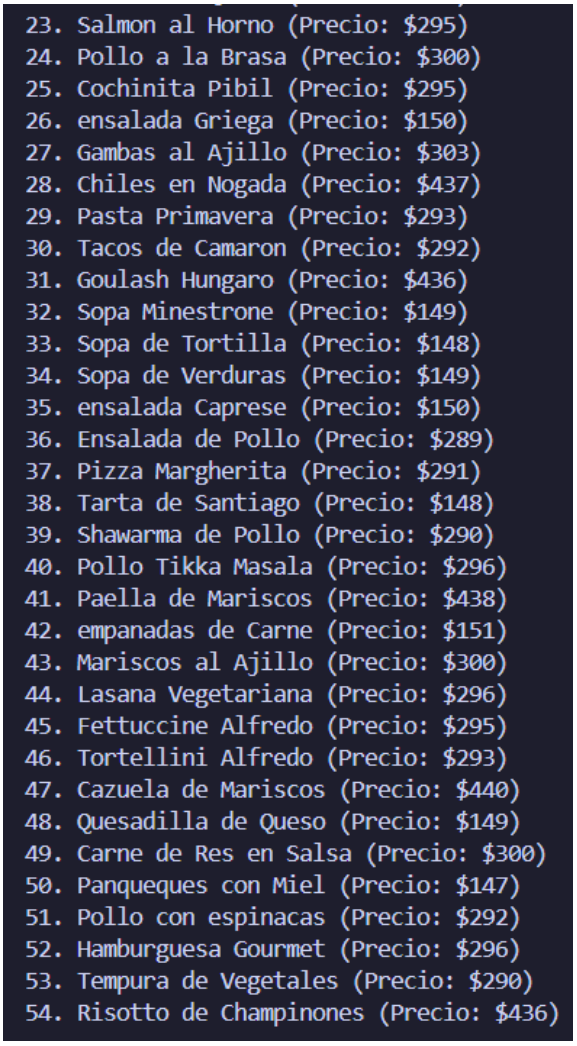
Búsqueda del menú de un restaurante

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

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

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

1. Pho Bo (Precio: $292)
2. Dim Sum (Precio: $295)
3. Chow Mein (Precio: $292)
4. Osso Buco (Precio: $445)
5. Chop Suey (Precio: $293)
6. Sushi Roll (Precio: $293)
7. Bruschetta (Precio: $146)
8. Arroz Frito (Precio: $289)
9. ensalada NiÔö£ŢÑoise (Precio: $288)
10. Mole Poblano (Precio: $295)
11. Ratatouille (Precio: $293)
12. Sopa de Pollo (Precio: $149)
13. Cordero Asado (Precio: $444)
14. Fajitas de Res (Precio: $298)
15. Pasta al Pesto (Precio: $294)
16. Pollo con Mole (Precio: $296)
17. Curry de Pollo (Precio: $296)
18. ensalada Cesar (Precio: $288)
19. Sopa de Pescado (Precio: $436)
20. Tacos de Birria (Precio: $292)
21. Tacos de Lengua (Precio: $292)
22. Fondue de Queso (Precio: $436)
```



Búsqueda de la existencia de un platillo en los restaurantes

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

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

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

1. Rios (Precio: $295)
2. Ocean (Precio: $256)
3. The Main (Precio: $299)
4. La Abuela (Precio: $282)
5. El Grande (Precio: $293)
6. El Pueblo (Precio: $278)
7. El Barzon (Precio: $276)
8. The Taste (Precio: $261)
9. El dragon (Precio: $267)
10. La Terraza (Precio: $258)
11. The Rustic (Precio: $301)
12. The Savory (Precio: $276)
13. The eatery (Precio: $273)
14. La Francesa (Precio: $261)
15. Bosque Real (Precio: $282)
16. The American (Precio: $271)
17. Los Rincones (Precio: $252)
18. Cocina Miguel (Precio: $278)
19. The Food Haven (Precio: $265)
20. La Rosticeria (Precio: $278)
21. The Trendy Cafe (Precio: $284)
22. The Garden Table (Precio: $284)
```

```
23. La Cocina de Mama (Precio: $261)
24. The Rustic Table (Precio: $258)
25. El Rincon del Mar (Precio: $261)
26. La Cocina del Chef (Precio: $261)
27. The Caffeine Spot (Precio: $250)
28. The Artisan Table (Precio: $254)
29. The Modern Palate (Precio: $273)
30. La Cantina de Juan (Precio: $284)
31. La Terraza del Mar (Precio: $288)
32. The Bistro Lounge (Precio: $293)
33. The Trendy eatery (Precio: $293)
34. La Taberna del Chef (Precio: $282)
35. El Cafe de la Abuela (Precio: $261)
36. La Terraza del Chef (Precio: $301)
37. La esquina del Chef (Precio: $256)
38. The elegant Bistro (Precio: $306)
39. La Parrilla Argentina (Precio: $295)
40. The Not Italian Bistro (Precio: $312)
41. El Rinconcito Tropical (Precio: $299)
42. The Mediterranean Table (Precio: $265)
43. La Brasserie Parisienne (Precio: $267)
44. El Restaurante de la Plaza (Precio: $280)
45. El Restaurante de la esquina (Precio: $293)
```

Restaurantes cercanos

```
Opciones
1- Busqueda del menu de un restaurante en especifico
2- Busqueda de restaurantes que ofrecen un plato en especifico
3- Continuar
> 3

Busqueda de restaurantes cercanos
Seleccione una posicion de inicio (formato: (0, 0)): (17, 26)

Restaurantes mas cercanos a la posicion (17, 26)
1. La Parrilla Argentina a 80 metros
2. Urban eats a 90 metros
3. The Countryside a 130 metros
```

Consulta de menú después de la búsqueda

```
Opciones
1- Busqueda del menu de un restaurante en especifico
2- Busqueda de restaurantes que ofrecen un plato en especifico
3- Continuar
> 3

Busqueda de restaurantes cercanos
Seleccione una posicion de inicio (formato: (0, 0)): (17, 26)

Restaurantes mas cercanos a la posicion (17, 26)
1. La Parrilla Argentina a 80 metros
2. Urban eats a 90 metros
3. The Countryside a 130 metros

Consultar menu de los restaurantes
1. Si
2. No
> 1
Seleccione un restaurante de los encontrados: 2

--------------------------------
Menu de Urban eats
--------------------------------

1. Fideua (Precio: $308)
2. Pad Thai (Precio: $308)
3. Osso Buco (Precio: $468)
4. Chop Suey (Precio: $308)
5. Churrasco (Precio: $461)
6. Pollo Kiev (Precio: $456)
7. Carne Asada (Precio: $314)
8. Pozole Rojo (Precio: $303)
9. Sopa de Miso (Precio: $155)
10. ensalada NiÔÖ£┬Ñoise (Precio: $303)
11. Lomo Saltado (Precio: $312)
```

```
12. Ratatouille (Precio: $308)
13. Costillas BBQ (Precio: $461)
14. Cordero Asado (Precio: $466)
15. Filete Mignon (Precio: $475)
16. Ceviche Mixto (Precio: $316)
17. Fajitas de Res (Precio: $313)
18. Pasta al Pesto (Precio: $309)
19. Pollo al Curry (Precio: $311)
20. ensalada Cesar (Precio: $303)
21. Sopa de Pescado (Precio: $458)
22. Tacos de Lengua (Precio: $307)
23. Sashimi de Atun (Precio: $311)
24. Pollo a la Brasa (Precio: $316)
25. Salmon al Horno (Precio: $310)
26. Tacos al Pastor (Precio: $157)
27. Lasana Bolonesa (Precio: $311)
28. Beef Stroganoff (Precio: $461)
29. Beef Wellington (Precio: $471)
30. Tacos de Camaron (Precio: $307)
31. Gnocchi al Pesto (Precio: $309)
32. Sopa Minestrone (Precio: $157)
33. Sopa de Tortilla (Precio: $155)
34. Sopa de Verduras (Precio: $156)
35. Tamales de elote (Precio: $155)
36. Cochinillo Asado (Precio: $474)
37. Gazpacho Andaluz (Precio: $156)
38. ensalada Caprese (Precio: $158)
39. Pizza Margherita (Precio: $305)
40. Yakitori de Pollo (Precio: $305)
41. Paella Valenciana (Precio: $316)
42. Pollo Tikka Masala (Precio: $311)
43. Chimichurri Steak (Precio: $458)
44. Tacos de Cochinita (Precio: $307)
45. Paella de Mariscos (Precio: $461)
46. Crepas con Nutella (Precio: $155)
47. Tournedos Rossini (Precio: $478)
48. Panuchos Yucatecos (Precio: $303)
```

```
49. Ravioles de Ricotta (Precio: $310)
50. Panqueques con Miel (Precio: $155)
51. estofado de Ternera (Precio: $313)
52. Pollo con espinacas (Precio: $307)
53. Spaghetti Carbonara (Precio: $310)
54. Bacalao a la Vizcaina (Precio: $459)
55. Crema de Champinones (Precio: $158)
56. Hamburguesa Gourmet (Precio: $311)
57. Pollo a la Cacciatora (Precio: $313)
58. Costillas de Cordero (Precio: $471)
59. Tempura de Vegetales (Precio: $305)
60. Raviolis de espinaca (Precio: $310)
61. Bratwurst con Chucrut (Precio: $309)
62. escargots a la Bourguignonne (Precio: $322)
```

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 5”): https://github.com/burnoz/situacion-problema