Tecnológico de Monterrey

**“Actividad Integral de arboles”**

**Programación de estructuras de datos y algoritmos fundamentales**

**Profesora: María Valentina Narváez Terán**

**Bruno Fernando Zabala Peña - A00838627**

**08/11/2024**

## 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 de resolución de la situación problema, se implementó el uso de árboles tipo heap (max) para organizar los restaurantes de acuerdo a sus ganancias y número de órdenes recibidas. Se eligió trabajar con este tipo de árbol debido a la facilidad con la que se pueden extraer los valores más grandes de un conjunto, además, las operaciones asociadas a estos son bastante eficientes, pues sus complejidades son las siguientes:

- Inserción (push): O(log(n))
- Pop: O(log(n))

- Heapify: O(log(n))
- Ordenamiento (pop hasta vaciar el heap): O(nlog(n))

Haciendo que esta estructura de datos sea más que apropiada para trabajar con un gran volumen de información.

Respecto a la etapa anterior, se mantuvo la implementación del quicksort con stack (O(nlog(n))) para ordenar las peticiones por orden alfabético, por lo que después de abrir el archivo, cada una de las entradas es almacenada en una lista enlazada doble, la cual es pasada a la función para su ordenamiento:

```
// Lista para almacenar los datos de las ordenes
List<string> datos;

// Pasa por cada linea del archivo
while(getline(file, line)){
    // Copia la linea al vector de ordenes
    // orders[i] = line;
    datos.insertLast(line);
    i++;
}

// Ordena las ordenes por restaurante
quickSortR(datos);
```

```cpp
// quicksort
// Ordena por restaurantes
// Complejidad: O(nlog(n))
template<class T>
void quickSortR(List<T> datos){
    Stack<Node<string>*> stack;

    Node<T> *L = datos.getFirst();
    Node<T> *R = datos.getLast();
    Node<T> *pivot = NULL;

    //  El primer par de L y R entran en la stack
    //  Esta particion inicial son el inicio y fin de la lista de datos
    stack.push(L);
    stack.push(R);

    //  En cada iteracion un par L y R salen de la stak
    //  La particion de L a R es reordenada
    //  Y segun donde quede el pivote, agrega nuevos pares de L y R a la stack
    while(stack.isEmpty() == false){
        R = stack.pop();
        L = stack.pop();

        pivot = partitionR(datos, L, R);

        if(pivot != L && pivot->next != L){
            stack.push(L);
            stack.push(pivot->prev);
        }

        if(pivot != R && pivot->prev != R){
            stack.push(pivot->next);
            stack.push(R);
        }

        // datos.showList();   // <---- Descomenta para ver el paso a paso
    }
}
```

```cpp
// Funcion para particionar la lista de ordenes (ordenar por restaurantes)
template<class T>
Node<T>* partitionR(List<T> datos, Node<T> *L, Node<T> *R){
    Node<T> *j = L;
    Node<T> *i = NULL;
    Node<T> *pivot = R;

    T aux = "";

    //  Compara con pivot e intercambia
    while(j != R){
        // cout << "\tComaparo " <<  j->value << " y " << pivot->value << endl;
        if(get_restaurant(j->value) < get_restaurant(pivot->value)){
            if(i == NULL){
                i = L;
            }

            else{
                i = i->next;
            }

            aux = j->value;
            j->value = i->value;
            i->value = aux;
        }

        j = j->next;
    }

    //  Coloca el valor pivote en su sitio
    if(i == NULL){
        i = L;
    }

    else{
        i = i->next;
    }

    aux = i->value;
    i->value = pivot->value;
    pivot->value = aux;

    return i;   // Devuelve la direccion de donde quedo el valor pivote
}
```

Para guardar la información de cada restaurante en el archivo, se implementó una clase que guarda el nombre del restaurante, el número de órdenes y sus ganancias:

```cpp
// Clase para restaurantes
class Restaurant{
    public:
        string name; // Nombre del restaurante
        int numOrders; // Numero de ordenes
        float revenue; // Ganancias

        // Constructor sin parametros
        Restaurant(){
            name = "";
            numOrders = 0;
            revenue = 0;
        }

        // Constructor con parametros
        Restaurant(string name, int numOrders, float revenue){
            this->name = name;
            this->numOrders = numOrders;
            this->revenue = revenue;
        }

        void show();
};
```

Para los heaps, estos se implementaron pasándole como atributos el número máximo de elementos, el número actual de elementos, un arreglo de objetos tipo Restaurant y un string que indica si este va a ser ordenado de acuerdo al número de órdenes o ganancias:

```cpp
// Clase para max heap de restaurantes
class Heap{
    public:
        Restaurant *restaurants; // Arreglo de restaurantes
        string type; // "revenue" o "numOrdenes"
        int maxSize; // Numero maximo de elementos en el heap
        int currentSize; // Numero actual de elementos

        // Constructor
        Heap(int maxSize, string type){
            this->maxSize = maxSize;
            this->currentSize = 0;
            this->type = type;
            this->restaurants = new Restaurant[maxSize];
        }

        // Destructor
        ~Heap(){
            delete[] restaurants;
            maxSize = 0;
            currentSize = 0;
        }

        // Funciones
        bool isEmpty();
        bool isFull();

        int parent(int);
        int left(int);
        int right(int);

        void push(Restaurant);
        Restaurant pop();
        void heapify(int);
        void swap(int, int);
        void show();
};
```

El atributo type es utilizado como variable de control en las funciones push y heapify (e indirectamente en pop) para indicarle la manera en que tienen que estar ordenados los restaurantes:

```cpp
// Inserta un elemento en el heap
// Complejidad: O(log(n))
void Heap::push(Restaurant restaurant){
    if(!isFull()){
        int i = currentSize;
        int p = parent(i);

        // Verifica si es un max heap por ganancia
        if(type == "revenue"){
            while(i > 0 && restaurant.revenue > restaurants[p].revenue){
                restaurants[i] = restaurants[p];
                i = p;
                p = parent(i);
            }
        }

        // Verifica si es un max heap por numero de ordenes
        else if(type == "numOrdenes"){
            while(i > 0 && restaurant.numOrders > restaurants[p].numOrders){
                restaurants[i] = restaurants[p];
                i = p;
                p = parent(i);
            }
        }

        // Inserta el elemento en el heap y aumenta el numero de elementos
        restaurants[i] = restaurant;
        currentSize++;
    }

    else{
        cout << "Heap lleno" << endl;
    }
}
```

```cpp
// Regresa el elemento superior del heap
// Complejidad: O(log(n)) por heapify
Restaurant Heap::pop(){
    if(!isEmpty()){
        // Guarda el elemento superior
        Restaurant top = restaurants[0];

        // Mueve el ultimo elemento al inicio
        restaurants[0] = restaurants[currentSize - 1];

        // Reduce el numero de elementos y reordena el heap
        currentSize--;
        heapify(0);

        // Regresa el elemento superior
        return top;
    }

    else{
        cout << "Heap vacio" << endl;
        return Restaurant();
    }
}
```

```cpp
// Reordena el heap
// Complejidad: O(log(n))
void Heap::heapify(int i){
    int l = left(i); // Indice del hijo izquierdo
    int r = right(i); // Indice del hijo derecho
    int largest = i; // Indice del elemento actual

    // Verifica si es un max heap por ganancia
    if(type == "revenue"){
        // Compara el elemento actual con su hijo izquierdo
        if(l < currentSize && restaurants[l].revenue > restaurants[largest].revenue){
            largest = l;
        }

        // Compara el elemento actual con su hijo derecho
        if(r < currentSize && restaurants[r].revenue > restaurants[largest].revenue){
            largest = r;
        }
    }

    // Verifica si es un max heap por numero de ordenes
    else if(type == "numOrdenes"){
        // Compara el elemento actual con su hijo izquierdo
        if(l < currentSize && restaurants[l].numOrders > restaurants[largest].numOrders){
            largest = l;
        }

        // Compara el elemento actual con su hijo derecho
        if(r < currentSize && restaurants[r].numOrders > restaurants[largest].numOrders){
            largest = r;
        }
    }

    // Si el elemento actual no es el mayor
    if(largest != i){
        // Intercambia el elemento actual con su hijo mayor
        swap(i, largest);
        heapify(largest);
    }
}
```

Con estas clases y la lista ya ordenada, el siguiente paso es crear los objetos para cada restaurante y añadirlos a los heaps correspondientes, lo cual se hace de la siguiente manera:

- Se crean dos heaps, uno para ordenar por ganancias y otro por número de órdenes.

```cpp
// Heaps para ordenar los restaurantes por numero de ordenes y ganancias
Heap heap_numOrders(10000, "numOrdenes");
Heap heap_revenue(10000, "revenue");
```

- Se inicializan variables para guardar la información del restaurante, así como un nodo auxiliar que guarda el primer elemento en la lista de datos.

```
// Variables para crear los objetos
string restaurant_name = "";
int numOrders = 0;
float revenue = 0;
Node<string> *aux = datos.getFirst();
```

- Se comienza a iterar en la lista, aumentando el número de órdenes y acumulando las ganancias del restaurante (este valor se extrae con la función get_revenue) en cada iteración.

```
// Pasa por cada orden y agrega el restaurante a los heaps
for(int j = 0; j < i; j++){
    // Aumenta el numero de ordenes del restaurante
    numOrders++;

    // Aumenta las ganancias del restaurante
    revenue += get_revenue(aux->value);
```

```
// 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);
}
```

- Si el restaurante en el elemento siguiente es distinto al actual, se procede a la creación del objeto del restaurante.
    - Se toma el nombre del restaurante con get_restaurant.
    - Se pasan los parámetros correspondientes al constructor del restaurante.
    - Se agrega el objeto a los heaps.

```
// Verifica si el restaurante del nodo actual no es igual al del nodo siguiente
if(get_restaurant(aux->value) != get_restaurant(aux->next->value)){
    // Nombre del restaurante
    restaurant_name = get_restaurant(aux->value);

    // Crea un objeto de tipo Restaurant
    Restaurant restaurant(restaurant_name, numOrders, revenue);

    // Inserta el restaurante en los heaps
    heap_numOrders.push(restaurant);
    heap_revenue.push(restaurant);

    // Reinicia las variables
    numOrders = 0;
    revenue = 0;
}
```

- Avanza en la lista

```
aux = aux->next;
```

Una vez que se guardó la información de todos los restaurantes, estos son almacenados de forma ordenada en archivos de texto de salida (se hace pop en los heaps hasta vaciarlos, O(nlog(n))) y se imprimen los primeros 10 elementos de cada heap en la consola:

```
// Guarda el contenido de los heaps en los archivos de salida
sorted_numOrders << "Lista de restaurantes por numero de ordenes" << endl;
cout << "Restaurantes con mayor numero de ordenes" << endl;
while(!heap_numOrders.isEmpty()){
    Restaurant restaurant = heap_numOrders.pop();
    num++;
    sorted_numOrders << num << "- " << restaurant.name << ": " << restaurant.numOrders << " ordenes, $" << restaurant.revenue << endl;

    // Imprime los primeros 10 restaurantes
    if(num <= 10){
        cout << num << "- " << restaurant.name << ": " << restaurant.numOrders << " ordenes, $" << restaurant.revenue << endl;
    }
}
```

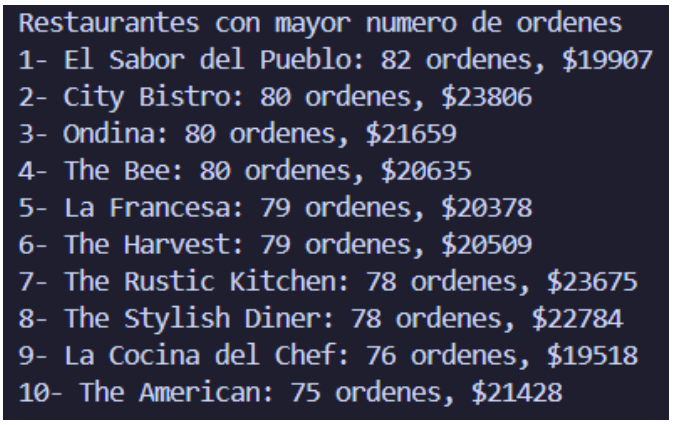
```
sorted_revenue << "Lista de restaurantes por ganancias" << endl;
cout << "Restaurantes con mayores ganancias" << endl;
while(!heap_revenue.isEmpty()){
    Restaurant restaurant = heap_revenue.pop();
    num++;
    sorted_revenue << num << "- " << restaurant.name << ": " << restaurant.numOrders << " ordenes, $" << restaurant.revenue << endl;

    // Imprime los primeros 10 restaurantes
    if(num <= 10){
        cout << num << "- " << restaurant.name << ": " << restaurant.numOrders << " ordenes, $" << restaurant.revenue << endl;
    }
}
```

Debido a los requerimientos establecidos para esta etapa, el código lee directamente la ruta del archivo orders.txt, además, se dejó la organización por

subcarpetas, por lo que los archivos de salida se crean en la misma ruta que el archivo de código, haciendo que su acceso pueda ser más rápido.

## Pruebas

### Ordenamiento por número de órdenes



main.cpp U  sorted_numOrders.txt U

Etapa 3 > sorted_numOrders.txt

```
Lista de restaurantes por numero de ordenes
1- El Sabor del Pueblo: 82 ordenes, $19907
2- City Bistro: 80 ordenes, $23806
3- Ondina: 80 ordenes, $21659
4- The Bee: 80 ordenes, $20635
5- La Francesa: 79 ordenes, $20378
6- The Harvest: 79 ordenes, $20509
7- The Rustic Kitchen: 78 ordenes, $23675
8- The Stylish Diner: 78 ordenes, $22784
9- La Cocina del Chef: 76 ordenes, $19518
10- The American: 75 ordenes, $21428
11- Urban eats: 74 ordenes, $23717
12- Sunset Grill: 74 ordenes, $20664
13- La Furia: 73 ordenes, $19979
14- El Sol: 73 ordenes, $20929
15- La Bella Notte: 73 ordenes, $20667
16- El Rincon del Gourmet: 72 ordenes, $18365
17- La Parrilla Argentina: 71 ordenes, $20428
18- El Sazon Mexicano: 71 ordenes, $19482
19- The Harvest House: 71 ordenes, $19864
20- The Rustic Spoon: 70 ordenes, $19376
21- Marthas: 70 ordenes, $18150
22- El Cafe de la Abuela: 70 ordenes, $18100
23- The Trendy eatery: 69 ordenes, $20438
24- Classic Cafe: 69 ordenes, $20926
25- El Restaurante de Ana: 69 ordenes, $19277
26- The Lounge: 69 ordenes, $18521
27- El Restaurante de la Plaza: 68 ordenes, $19556
28- La esquina del Taco: 68 ordenes, $16768
29- El Marisco Feliz: 68 ordenes, $18017
30- The Bistro Lounge: 68 ordenes, $21210
31- The Cafe Lounge: 68 ordenes, $18698
32- The Not Italian Bistro: 67 ordenes, $20135
33- The Savory: 67 ordenes, $19147
34- The Trendy Cafe: 67 ordenes, $17724
35- Las salsas: 67 ordenes, $19866
36- Los Rincones: 67 ordenes, $15638
```

## Ordenamiento por ganancias

```
Restaurantes con mayores ganancias
1- City Bistro: 80 ordenes, $23806
2- Urban eats: 74 ordenes, $23717
3- The Rustic Kitchen: 78 ordenes, $23675
4- The Stylish Diner: 78 ordenes, $22784
5- Ondina: 80 ordenes, $21659
6- The American: 75 ordenes, $21428
7- The Bistro Lounge: 68 ordenes, $21210
8- El Sol: 73 ordenes, $20929
9- Classic Cafe: 69 ordenes, $20926
10- La Bella Notte: 73 ordenes, $20667
```


main.cpp U
sorted_revenue.txt U
Etapa 3 > sorted_revenue.txt
Lista de restaurantes por ganancias
1- City Bistro: 80 ordenes, $23806
2- Urban eats: 74 ordenes, $23717
3- The Rustic Kitchen: 78 ordenes, $23675
4- The Stylish Diner: 78 ordenes, $22784
5- Ondina: 80 ordenes, $21659
6- The American: 75 ordenes, $21428
7- The Bistro Lounge: 68 ordenes, $21210
8- El Sol: 73 ordenes, $20929
9- Classic Cafe: 69 ordenes, $20926
10- La Bella Notte: 73 ordenes, $20667
11- Sunset Grill: 74 ordenes, $20664
12- The Bee: 80 ordenes, $20635
13- The Spot: 64 ordenes, $20588
14- The Harvest: 79 ordenes, $20509
15- The Trendy eatery: 69 ordenes, $20438
16- La Parrilla Argentina: 71 ordenes, $20428
17- La Francesa: 79 ordenes, $20378
18- The elegant Bistro: 65 ordenes, $20342
19- The Not Italian Bistro: 67 ordenes, $20135
20- La Furia: 73 ordenes, $19979
21- La Cocina de Alicia: 65 ordenes, $19940
22- El Sabor del Pueblo: 82 ordenes, $19907
23- Las salsas: 67 ordenes, $19866
24- The Harvest House: 71 ordenes, $19864
25- The Village: 62 ordenes, $19798
26- El Restaurante de la Plaza: 68 ordenes, $19556
27- La Cocina del Chef: 76 ordenes, $19518
28- El Sazon Mexicano: 71 ordenes, $19482
29- Brunch & Mimosas: 65 ordenes, $19434
30- Casa de las empanadas: 64 ordenes, $19411
31- The Rustic Spoon: 70 ordenes, $19376
32- Here and Now: 66 ordenes, $19287
33- El Restaurante de Ana: 69 ordenes, $19277
34- El Rincon del Chef: 64 ordenes, $19253
35- The Classic: 60 ordenes, $19241
36- The Food Haven: 66 ordenes, $19187

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