Mini-Proyecto Nº 2: QuadTree

INTEGRANTES Ríos Adasme, Vicente Ignacio, 2020449783
San Martín Vargas, Joaquín Ignacio, 2021447687
Sanhueza Yévenes, Pablo Alonso, 2021439005

PROFESOR José Sebastián Fuentes Sepúlveda

Martes, 27 de junio de 2023, Concepción

Índice

1. Introducción

El tema de este proyecto se centra en la implementación de un QuadTree, una estructura de datos diseñada para realizar búsquedas en claves bidimensionales en un plano 2D. Se utiliza el ejemplo de un mapa o plano en el cual están marcadas diferentes ciudades, donde cada ciudad puede ser identificada de manera única mediante sus coordenadas (x, y). El objetivo del proyecto es utilizar el DataSet "World Cities Population", que contiene información sobre más de 3 millones de ciudades, y utilizar las coordenadas de cada ciudad como puntos en el plano 2D del QuadTree. Además, se debe almacenar al menos la población de cada ciudad.

El QuadTree permite representar puntos en el plano 2D de manera eficiente, evitando el uso de espacio en zonas del plano que no contienen puntos. Para construir el QuadTree, se comienza representando todo el plano como un solo cuadrante y se divide recursivamente en cuatro cuadrantes más pequeños, clasificándolos como nodos blancos o negros dependiendo de si contienen puntos o no.

En términos de implementación, se solicita crear una clase llamada QuadTree que tenga métodos para realizar diferentes operaciones, como la inserción de puntos, el cálculo del número total de puntos y nodos, la obtención de una lista de todos los puntos almacenados y el conteo y agregación de puntos en una región del plano.

El proyecto se enfoca en aplicar los conceptos del QuadTree para representar puntos en un plano 2D utilizando un dataset de ciudades, y se requiere implementar métodos para realizar diversas operaciones en el QuadTree.

2. Descripción de QuadTree

De Hanan Samet [1] se tiene que QuadTree es una estructura de datos jerárquica utilizada para organizar y buscar información espacial en un plano bidimensional. Su nombre proviene de la subdivisión recursiva del espacio en cuatro regiones cuadradas más pequeñas, conocidas como cuadrantes. Cada nodo del QuadTree representa un cuadrante y puede tener hasta cuatro hijos, que corresponden a los cuatro cuadrantes resultantes de la subdivisión. Esta subdivisión se realiza de manera recursiva hasta alcanzar un nivel deseado de detalle o hasta cumplir un criterio de parada.

La principal ventaja del QuadTree radica en su capacidad para representar puntos en un plano 2D de manera eficiente, evitando el gasto de espacio en zonas que no contienen puntos. Los nodos del QuadTree pueden ser de dos tipos: blancos o negros. Los nodos blancos indican que el cuadrante que representan no contiene puntos, mientras que los nodos negros indican que el cuadrante contiene al menos un punto. Esto permite una representación compacta de la información espacial.

El QuadTree se utiliza ampliamente en aplicaciones que involucran datos espaciales, como sistemas de información geográfica, gráficos por computadora, colisiones en juegos y algoritmos de búsqueda espacial. Permite realizar diversas operaciones sobre los datos, como la inserción de nuevos puntos, la búsqueda de puntos en una región específica del plano, el recuento de puntos en una región y la agregación de valores asociados a los puntos, como la población de una ciudad.

La construcción de un QuadTree comienza con un cuadrante que representa todo el plano. Luego, se divide recursivamente en cuatro cuadrantes más pequeños, clasificándolos como nodos blancos o negros según la presencia de puntos. Esta subdivisión continúa hasta llegar a celdas individuales, donde se almacena la información asociada a cada punto. Todos los nodos internos del QuadTree son negros, mientras que las hojas pueden ser blancas o negras.

3. Implementación propuesta

La descripción anterior corresponde a la descripción habitual de un Quadtree, esta estructura de datos espaciales se utiliza para dividir el espacio en cuadrantes y organizar los puntos en función de su ubicación. La implementación propuesta sigue este enfoque general y utiliza técnicas recursivas para realizar operaciones como la inserción, el recuento de puntos y el recorrido del árbol.

Es importante destacar que la implementación no se basa en un autor en específico o reconocido. En cambio, se ha desarrollado utilizando el conocimiento comúnmente aceptado sobre Quadtree. Si bien existen diferentes variaciones y enfoques para implementar un Quadtree, esta implementación sigue las convenciones generales de la estructura de datos.

Cabe destacar, que nuestra implementación posee dos métodos de los cuales no se nos solicita implementar, sin embargo, son útiles para obtener un mejor manejo de la información almacenada en la estructura. Eso sí, en la descripción de la implementación, es decir, a continuación, se será explícito en mostrar cuales son dichos métodos.

Toda la documentación e implementación realizada se encuentra en el repositorio en el cual se puede acceder en este enlace.

3.1 Archivos de encabezado (.hpp)

En C++, es común seguir una convención en la cual se comienza describiendo los archivos de cabecera (header files) antes de proporcionar la implementación de los métodos en los archivos de código fuente (.cpp). Estructura que se ha aprendido y se ha mantenido en el ejercicio práctico durante la formulación de laboratorios de la presente asignatura. Esta práctica ayuda a establecer una interfaz clara y separar la declaración de los elementos de su implementación.

3.1.1 Node

Se tiene Node, que representa un nodo en el QuadTree. La estructura Node tiene los siguientes atributos:

I. "point": Representa las coordenadas del punto asociado al nodo.
II. "cityName": Almacena el nombre de la ciudad asociada al nodo.
III. "population": Indica la población de la ciudad asociada al nodo.
IV. "color": Es una cadena de texto que indica el color del nodo. Puede ser "Black" para representar un nodo con información o "White" para representar un nodo vacío.
V. "NW", "NE", "SW", "SE": Son punteros a los nodos hijos del nodo actual, que corresponden a los cuadrantes noroeste, noreste, suroeste y sureste, respectivamente.

La implementación de la estructura Node proporciona dos constructores. El primer constructor acepta los parámetros de punto, nombre de la ciudad, población y color, y se utiliza para crear un nodo con información. El segundo constructor no recibe parámetros y se utiliza para generar un nodo vacío disponible, estableciendo el color como "White".

La estructura Node encapsula los datos asociados a cada nodo en el QuadTree y proporciona una forma de almacenar información geoespacial relacionada con ciudades y su población.

3.1.2 Point

Se define la estructura de datos Point, que representa un punto en un plano bidimensional. La estructura Point tiene los siguientes atributos:

I. "x": Representa la coordenada x del punto.
II. "y": Representa la coordenada y del punto.

La implementación de la estructura Point proporciona dos constructores. El primer constructor acepta los parámetros de las coordenadas x e y, y se utiliza para crear un punto con valores específicos. El segundo constructor no recibe parámetros y se utiliza para generar un punto con coordenadas predeterminadas, estableciendo tanto x como y en 0.

La estructura Point es útil para almacenar y manipular información relacionada con las coordenadas de puntos en un plano bidimensional, como se utiliza en el contexto del QuadTree para representar ubicaciones geoespaciales.

3.1.3 QuadTree

Se proporciona la declaración de la clase QuadTree, incluyendo sus atributos y métodos públicos y privados. En cuanto a los atributos de la clase QuadTree, se tienen dos:

I. "root": Es un puntero a Node que representa la raíz del QuadTree. Este atributo es privado y se utiliza para acceder al nodo raíz del árbol.

II. "insertCount": Es un entero que registra el número de inserciones realizadas en el QuadTree. Este atributo también es privado y se utiliza para llevar un seguimiento del número de puntos insertados en el árbol.

Estos atributos son parte de la implementación interna de la clase QuadTree y son utilizados para gestionar la estructura y el comportamiento del árbol.

3.2 Implementación métodos

3.2.1 totalPoints()

Esta función devuelve el valor actual de la variable entera insertCount, la cual se incrementa cada vez que se llama al método insert. Es decir, totalPoints retorna la cantidad total de puntos que se han insertado en el QuadTree hasta el momento de su llamada.

3.2.2 totalNodes()

Para la implementación del método totalNodes se utiliza la función totalNodesRecursive, que es una función privada y se ejecuta de manera recursiva.

En la función totalNodes, se inicializa un contador en 0. Luego, se llama a totalNodesRecursive pasando como argumentos el puntero a la raíz del QuadTree y el contador. Esta función recursiva se encarga de contar los nodos en el QuadTree.

En cada llamada recursiva, se verifica si el nodo actual no es nulo. Si el nodo no es nulo, se incrementa el contador en 1. A continuación, se realiza una llamada recursiva a la función totalNodesRecursive para cada uno de los cuatro hijos del nodo actual. La recursión continúa hasta que se haya contado todos los nodos en el QuadTree.

Al finalizar la recursión, se devuelve el valor del contador, que representa la cantidad total de nodos en el QuadTree.

3.2.3 insert(Point p, std::string cityName, int population)

El método insert en la clase QuadTree se encarga de insertar un nuevo punto en el QuadTree, junto con su nombre de ciudad y población asociada. El proceso de inserción se realiza de manera recursiva desde la raíz del QuadTree.

En el método insert, se invoca la función insertRecursive, pasando como argumentos la raíz del QuadTree, el punto a insertar, el nombre de la ciudad y la población. Esta función recursiva se encarga de realizar la inserción de manera adecuada en el QuadTree.

Primero, se verifica si el nodo actual es un nodo vacío. Si es así, se realiza la inserción en este nodo. Se asignan los valores correspondientes al nodo actual, como el color ("Black"), el punto, el nombre de la ciudad y la población. Luego, se crean nuevos nodos blancos para los hijos del nodo actual (NW, NE, SW, SE), si es que alguno de ellos es nulo.

Si el nodo actual contiene información, lo que indica que no es un nodo vacío, se divide el espacio y se determina en qué cuadrante cae el punto a insertar. Se calcula el punto medio del nodo actual y se compara la posición del punto con respecto a este punto medio para determinar en qué cuadrante se encuentra. A partir de esto, se selecciona el puntero al hijo correspondiente (NW, NE, SW, SE).

Si el hijo correspondiente no existe, es decir, el puntero es nulo, se crea un nuevo nodo vacío y se lo asigna como hijo. Luego, se realiza una llamada recursiva al método insertRecursive en el hijo correspondiente, pasando los mismos parámetros.

De esta manera, el proceso de inserción continúa de manera recursiva descendiendo en el QuadTree hasta encontrar un nodo vacío donde se pueda realizar la inserción. El proceso de inserción continúa de manera recursiva descendiendo por el QuadTree según las condiciones establecidas por las comparaciones de puntos. Esto asegura que el nuevo punto se inserte en la posición correcta del QuadTree.

Al finalizar la inserción, se incrementa el valor de insertCount en 1, lo que indica que se ha realizado una inserción exitosa en el QuadTree. El propósito de insertCount es mantener un registro de la cantidad de puntos insertados en el QuadTree.

3.2.4 getPointList()

Se decidió que el método getPointList() devuelve una lista de punteros a nodos que representan los puntos almacenados en el QuadTree.

Dentro del método, se crea una lista llamada pointList que se utilizará para almacenar los punteros a los nodos con información. Luego, se realiza una llamada a la función getPointListRecursive, pasando la raíz del QuadTree y la lista pointList.

La función getPointListRecursive es la encargada de realizar la recolección de los nodos con información. Toma como parámetros un puntero a un nodo y una referencia a la lista pointList.

En la función, se verifica si el nodo actual es de color "Black", lo que indica que contiene información. Si todos los hijos del nodo actual son de color "White" (nodos vacíos), se agrega el puntero al nodo actual a la lista pointList. Esto significa que el nodo actual es un nodo terminal en el QuadTree y contiene información relevante.

En caso contrario, si al menos uno de los hijos del nodo actual no es de color "White", se realiza una llamada recursiva a la función getPointListRecursive para cada uno de los cuatro hijos del nodo actual (NW, NE, SW, SE). Esto permite explorar todos los nodos descendientes del QuadTree en busca de aquellos que contienen información.

3.2.5 countRegion(Point p, int d)

Se cuenta recursivamente los puntos dentro de una región circular en base a un punto p y un radio d. Utiliza la función countRegionRecursive para realizar el conteo de puntos. La función countRegionRecursive verifica si el nodo actual no contiene puntos y devuelve 0, o si contiene puntos y calcula la distancia al cuadrado entre el punto actual y el punto p. Luego, realiza llamadas recursivas a la función para explorar los descendientes del QuadTree y contar los puntos dentro de la región.

En detalle, el método countRegion llama a la función countRegionRecursive, pasando la raíz del QuadTree, el punto p y el radio d. En la función countRegionRecursive, se verifica si el color del nodo actual es "White" (blanco). Esto indica que el nodo no contiene puntos y, por lo tanto, se devuelve 0.

Si el nodo actual contiene información (color "Black"), se calcula la distancia al cuadrado entre el punto almacenado en el nodo actual y el punto p. Esto se realiza mediante las diferencias en coordenadas dx y dy. A continuación, se inicializa una variable count en 0, que se utilizará para contar los puntos dentro de la región.

Si la distancia al cuadrado es menor o igual al cuadrado $d \cdot d$, significa que el punto se encuentra dentro de la región circular. En ese caso, se incrementa count en 1.

Luego, se realiza una llamada recursiva a la función countRegionRecursive para cada uno de los cuatro hijos del nodo actual (NW, NE, SW, SE), pasando el mismo punto p y radio d. Esto permite explorar todos los nodos descendientes del QuadTree.

Finalmente, se devuelve el valor de count, que representa el total de puntos contados dentro de la región circular. Este corresponde a una variable del tipo long long para almacenar una población adecuada para el DataSet, es decir, para guardar así, de manera más optima, cantidades superiores a las que permite un int en memoria (32 bits).

3.2.6 aggregateRegion(Point p, int d)

Análogo a countRegion, aggregateRegion calcula recursivamente la agregación de población dentro de una región circular en base a un punto p y un radio d. Utiliza la función aggregateRegionRecursive para realizar el cálculo de la agregación. La función aggregateRegionRecursive verifica si el nodo actual no contiene datos y devuelve 0 como valor agregado, o si contiene datos y calcula la distancia al cuadrado entre el punto actual y el punto p. Luego, realiza llamadas recursivas a la función para explorar los descendientes del QuadTree y sumar los valores de población dentro de la región. El resultado final es el valor agregado de población en la región circular.

Este método llama a aggregateRegionRecursive, pasando la raíz del QuadTree, el punto p y el radio d. En la función aggregateRegionRecursive, se verifica si el color del nodo actual es

"White" (blanco). Esto indica que el nodo no contiene datos y, por lo tanto, se devuelve 0 como valor agregado.

Si el nodo actual contiene información (color "Black"), se calcula la distancia al cuadrado entre el punto almacenado en el nodo actual y el punto p. Esto se realiza mediante las diferencias en coordenadas dx y dy. Luego, se inicializa una variable aggregate en 0, que se utilizará para almacenar el valor agregado de población.

Si la distancia al cuadrado es menor o igual al cuadrado $d \cdot d$, significa que el punto se encuentra dentro de la región circular. En ese caso, se agrega el valor de población del nodo actual al valor agregado (aggregate).

Luego, se realiza una llamada recursiva a la función aggregateRegionRecursive para cada uno de los cuatro hijos del nodo actual (NW, NE, SW, SE), pasando el mismo punto p y radio d. Esto permite explorar todos los nodos descendientes del QuadTree y sumar sus valores de población dentro de la región.

Finalmente, se devuelve el valor de aggregate, que representa la suma de los valores de población agregados dentro de la región circular. A igual que en el método anterior, este valor corresponde a una variable del tipo long long para almacenar una población adecuada para el DataSet.

3.2.7 clear()

El método clear en la clase QuadTree elimina todos los nodos del QuadTree y restablece su estado inicial. Primero, se llama a clearRecursive, que elimina recursivamente los nodos hijos y libera la memoria. Luego, se crea un nuevo nodo vacío que se asigna como la nueva raíz del QuadTree. Esto garantiza que el QuadTree esté vacío y listo para ser utilizado nuevamente.

La función clearRecursive es responsable de eliminar recursivamente los nodos del QuadTree. Si un nodo es de color "White", que indica que no contiene información ni hijos, se retorna. Si el nodo contiene información, se llama recursivamente a clearRecursive para cada uno de sus hijos en las cuatro direcciones. Finalmente, se libera la memoria del nodo actual mediante delete. En resumen, el método clear asegura que el QuadTree esté limpio y vacío, mientras que clearRecursive se encarga de eliminar los nodos de manera recursiva.

No que clear corresponde a un método que no se pide implementar explícitamente en el enunciado de este mini-proyecto.

3.2.8 printTotalPopulation ()

Este se encarga de imprimir la población total almacenada en todo el Quadtree. Primero, se llama a la función auxiliar calculateTotalPopulation() pasando como parámetro la raíz del Quadtree. Esta función realiza un recorrido recursivo por todos los nodos del Quadtree.

En la función calculateTotalPopulation(), se verifica si el color del nodo actual es "White" (blanco). Si es así, significa que el nodo no contiene datos y se devuelve 0 como población. En caso contrario, se inicializa la variable totalPopulation con la población del nodo actual.

Luego, se realiza la suma recursiva de la población de los hijos del nodo actual. Se llama a la función calculateTotalPopulation() para cada uno de los cuatro hijos (NW, NE, SW, SE) y se acumula la población retornada en la variable totalPopulation.

Note que, al igual que clear, printTotalPopulation corresponde a un método que no se pide implementar explícitamente en el enunciado de este mini-proyecto.

4. Análisis teórico

A continuación, se realiza análisis teórico de los métodos mencionados en el punto anterior. Los siguientes resultados son para el peor de los casos utilizando la notación asintótica Big-Oh vista en clases. Se tiene que 'n' es el número total de nodos en el QuadTree.

4.2 totalPoints()

Dado que retorna el valor de insertCount, que se incrementa cada vez que se llama al método insert. Por lo tanto, la complejidad de tiempo en el peor de los casos es O(1), ya que la operación es de tiempo constante.

4.3 totalNodes()

En el peor escenario posible, este método debe recorrer todos los nodos del QuadTree para contar el número total de nodos. Dado que cada nodo se visita una vez, la complejidad de tiempo en el peor de los casos es O(n).

4.4 insert(Point p, std::string cityName, int population)

Este método debe encontrar el lugar adecuado para insertar el punto en el QuadTree. Si el árbol está equilibrado, la complejidad de tiempo en el peor de los casos será O(log n). Sin embargo, en el peor de los casos, si el árbol está desequilibrado y forma una lista enlazada, la complejidad de tiempo puede ser O(n).

Con respecto a lo último, la forma se define según el orden de inserción de los puntos. Si los puntos se insertan en un orden aleatorio o no específico, es menos probable que se forme una lista enlazada. Sin embargo, si los puntos se insertan en un orden que siempre elige el mismo cuadrante en cada nivel, es posible que se forme una lista enlazada.

4.5 getPointList()

Este método debe recorrer todos los nodos del QuadTree para obtener una lista de puntos. Dado que se deben visitar todos los nodos, la complejidad de tiempo en el peor de los casos es O(n).

4.6 countRegion(Point p, int d)

Este método debe recorrer todos los nodos del QuadTree para contar los puntos dentro de una región. Dado que se deben visitar todos los nodos, la complejidad de tiempo en el peor de los casos es O(n).

4.7 aggregateRegion(Point p, int d)

En el peor de los casos, este método debe recorrer todos los nodos del QuadTree para agregar la población de los puntos dentro de una región. Dado que se deben visitar todos los nodos, la complejidad de tiempo en el peor de los casos es O(n).

4.8 clear()

Este método debe recorrer todos los nodos del QuadTree para eliminarlos y liberar la memoria. Dado que se deben visitar todos los nodos una vez, la complejidad de tiempo en el peor de los casos es O(n).

4.9 printTotalPopulation()

Evidentemente, se tiene que este método posee una complejidad de O(n). Esto se debe a que la función calculateTotalPopulation realiza un recorrido recursivo por todos los nodos del Quadtree.

5. Análisis experimental

5.1 Descripción relevante para análisis experimental

De lo aprendido en clases, se sabe que en un análisis experimental es fundamental tener en cuenta los detalles técnicos importantes de la máquina utilizada. Estos detalles proporcionan información esencial sobre el hardware y el entorno de ejecución, lo cual puede tener un impacto significativo en los resultados obtenidos.

En este caso, se utilizó un OMEN Laptop 15-ek0xxx con sistema operativo Microsoft Windows 11 Home. La CPU empleada fue un Intel(R) Core(TM) i5-10300H con una velocidad de reloj de 2496MHz. En el aspecto gráfico, la maquina cuenta con una GPU Intel(R) UHD Graphics y una NVIDIA GeForce RTX 2060. Además, se dispuso de una memoria RAM de 8GB con tecnología DDR4.

Estos detalles técnicos son relevantes porque diferentes configuraciones de hardware pueden afectar el rendimiento y la capacidad de procesamiento de la máquina. Por ejemplo, la velocidad de la CPU y la capacidad de la GPU pueden influir en la velocidad de cálculo. La cantidad de memoria RAM disponible también puede ser determinante para ejecutar aplicaciones y procesos de manera eficiente.

En este estudio, se evalúa el rendimiento de la implementación de QuadTree mediante la medición del tiempo requerido para realizar operaciones de inserción de ciudades obtenidas del DataSet mencionado en la introducción, cuyo documento es “worldcitiespop_fixed.csv”, y el tiempo que se tarda en reportar todas las ciudades y población en una determinada área. Se mide el tiempo total para insertar diferentes cantidades de ciudades (reportando tiempo promedio solo para la inserción), así como el tiempo requerido para reportar ciudades dentro de áreas de distintas dimensiones utilizando los métodos countRegion y aggregateRegion.

Se llevan a cabo múltiples pruebas utilizando diferentes cantidades de ciudades y dimensiones de área. En el caso del método insert, se evalúa el tiempo necesario para insertar conjuntos de ciudades de diferentes tamaños, estos son 100000, 200000, 400000, 800000, 1600000 y todas las ciudades disponibles, es decir, 3173647 ciudades. En el caso de los métodos countRegion y aggregateRegion, se realizan pruebas para contar se prueban con los siguientes radios: 15, 30, 45, 60, 75, 90.

Se repite este proceso un total de veinte veces, donde resultados de las mediciones se encuentran a continuación.

5.2 Resultados

Por temas de interpretación de resultados se crean dos archivos con extensión “.cpp” donde se realizan las pruebas, estos son “mainMétodos.cpp” y “mainInsert.cpp”. Cabe destacar que ambos scripts trabajan el QuadTree de la misma manera, solamente difieren en el uso de std::cout, donde el primero imprime los tiempos de los métodos countRegion y aggregateRegion, mientras que el segundo imprime los tiempos de las inserciones, ambos medidos en milisegundos.

La salida obtenida de los veinte experimentos de los métodos countRegion y aggregateRegion se encuentran en el archivo de texto “TestMetodos.txt” ubicado en el repositorio ya señalado. En la Tabla 1 se tiene los resultados del primer experimento.

Asimismo, en la Figura 1 se aprecia el tiempo, de la inserción de todas las posibles ciudades descritas anteriormente. La salida de las veinte experimentaciones se encuentra en “TestInsert.txt”.

Tabla 1: Comparación de countRegion y aggregateRegion en primer experimento

Ciudades	Radio	countRegion	Tiempo total countRegion (ms)	aggregateRegion	Tiempo total aggregateRegion (ms)
100000	15	4108	20.9283	991345696	19.5446
100000	30	8517	19.9056	2056839120	19.3457
100000	45	16659	19.184	-273840337	18.9307
100000	60	40102	18.7942	1110332325	19.25
100000	75	55871	19.8557	595573014	18.9814
100000	90	67638	23.1867	-869401652	18.7485
200000	15	12250	86.8665	-1336089235	87.9871
200000	30	25700	84.7548	1926155311	95.2046
200000	45	49967	78.0568	-813857851	63.2121
200000	60	120476	74.8373	-962752775	66.8675
200000	75	167484	65.3751	1772730429	72.1629
200000	90	202892	91.0419	1684644483	74.8256
400000	15	28485	201.235	-1663451714	167.525
400000	30	59881	168.012	1647335917	180.291
400000	45	116250	174.562	-1966268017	178.256
400000	60	281026	182.257	-912778055	183.91
400000	75	390498	180.605	-235095949	187.593
400000	90	473119	166.517	-1908690471	174.641
800000	15	61099	449.259	2071076523	477.673
800000	30	128482	435.189	1218267417	471.02
800000	45	248909	465.834	176165957	532.606
800000	60	602693	549.892	-511466553	502.799
800000	75	836328	466.939	131748378	496.04
800000	90	1013597	462.652	-260878569	467.963
1600000	15	125980	1084.69	808310751	1085.15
1600000	30	264818	993.673	130222243	909.791
1600000	45	513322	920.249	-38075806	936.649
1600000	60	1245389	900.444	134099467	913.501
1600000	75	1726605	902.457	534306563	869.959
1600000	90	2094050	882.034	-1233060224	893.386
3173647	15	128977	955.323	1517368162	941.124
3173647	30	271054	993.048	1745906494	951.18
3173647	45	525549	948.985	-1109228544	956.192
3173647	60	1273358	983.059	-1671682234	944.298
3173647	75	1765804	957.222	1484249221	956.307
3173647	90	2142874	929.098	2083914284	953.656

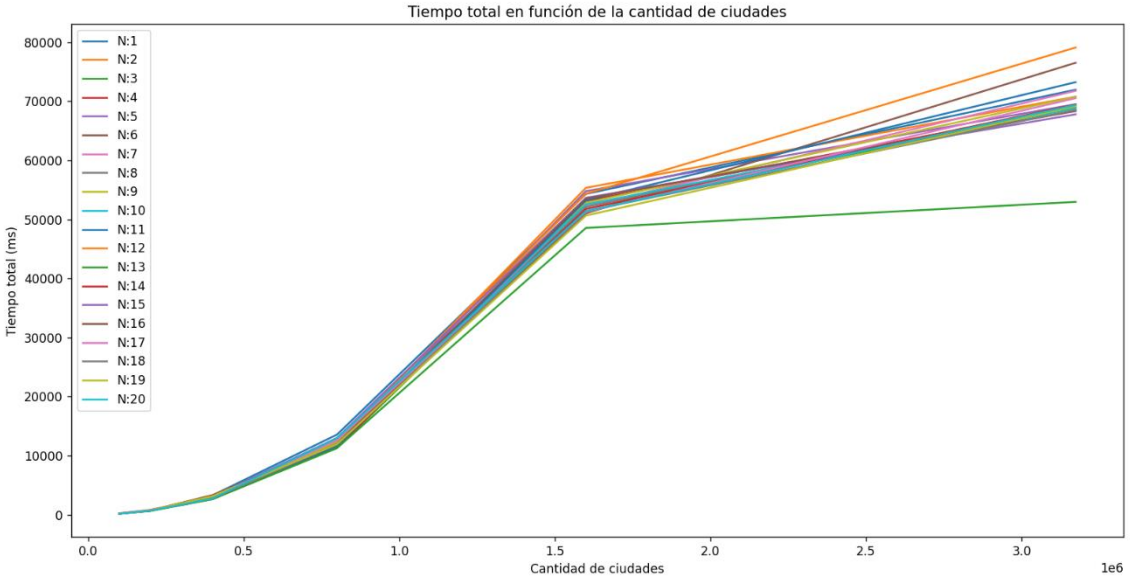

Figura 1: Comparación de los veinte experimentos para la inserción

6. Conclusiones

La existencia de estructuras de datos espaciales, como el Quadtree, representa una herramienta invaluable para el manejo eficiente de datos en un espacio bidimensional. A diferencia de las estructuras de datos convencionales vistas en clases, que se enfocan principalmente en la organización y acceso, muchas veces de manera secuencial, de datos, las estructuras de datos espaciales aprovechan la estructura geométrica del espacio para optimizar operaciones espaciales como búsqueda, conteo y agregación de puntos.

El uso de un árbol que representa el plano 2D, como el Quadtree, permite dividir el espacio en regiones más pequeñas lo que facilita la búsqueda rápida y eficiente de puntos cercanos o dentro de regiones específicas. Esto es especialmente valioso en aplicaciones donde la ubicación y la relación espacial de los datos son fundamentales, como sistemas de información geográfica, gráficos por computadora, detección de colisiones y muchas otras.

Teniendo en cuenta la representación espacial de esta estructura de datos, se pueden sacar algunas conclusiones con respecto a los resultados del análisis teórico realizado anteriormente. Es decir, al momento de realizar una comparación de eficiencia en estructuras de datos se debe considerar varios aspectos, por lo que depende del escenario de uso especifico y de las operaciones que se realizan con los datos definir qué estructura se considera más “eficiente”.

Dado que un QuadTree es especialmente útil cuando se trabaja con datos espaciales, como puntos en un plano. Su estructura jerárquica y la partición espacial que ofrece permiten realizar operaciones de búsqueda y consulta espacial de manera eficiente. Esto significa que, en general, el QuadTree puede ofrecer un mejor rendimiento en términos de tiempo de ejecución para operaciones como búsqueda de puntos en una región, recuento de puntos en una región, agregación de valores asociados a puntos, entre otros.

Sin embargo, cuando se trata de operaciones como acceso aleatorio por índice o inserción y eliminación de elementos en posiciones específicas, las estructuras de datos convencionales como arreglos, vectores o listas pueden ser más eficientes. Estas estructuras brindan un acceso directo a los elementos a través de su posición en el contenedor y suelen ser más eficientes en términos de tiempo de ejecución para estas operaciones.

Con respecto a la Tabla 1, se puede concluir que los datos obtenidos muestran que el tiempo de ejecución aumenta a medida que se incrementa la cantidad de ciudades y el tamaño del radio en ambas funciones, countRegion() y aggregateRegion(). También se observa que en algunos casos la estimación de población en la región muestra valores negativos. Además, se aprecia que las diferencias en los tiempos de ejecución entre las dos funciones son mínimas. En general, el número de puntos en la región aumenta a medida que se incrementa el radio.

De la Figura 1 se concluye que, como naturalmente se espera, el tiempo total en milisegundos aumenta a medida que se incrementa la cantidad de ciudades en los diferentes experimentos.

Existe una relación positiva entre la cantidad de ciudades y el tiempo requerido, aunque el crecimiento no es lineal. Se observa variabilidad en los tiempos totales para una misma cantidad de ciudades, lo cual puede deberse a diferentes condiciones de ejecución. No se puede establecer una tendencia clara en la forma de la curva de crecimiento, lo que sugiere una relación compleja y dependiente de otros factores. Es posible que para obtener resultados más precisos, se requiere un análisis más detallado y considerar otros factores específicos del experimento.

7. Referencias

[1] H. Samet, The Design and Analysis of Spatial Data Structures, Addison-Wesley, 1990.