The Introduction and Application of Quadtree

資管一 B10705005 陳思如

I. Introduction

i、 Basic Introduction

Quadtree is a tree data structure and has at most four children node. It is an extension of binary tree. Most of the time, people use quadtree to partition two-dimensional space by subdividing the regions into four quadrants because it has four children nodes, the classification can have four categories for left, right, up, and down. Due to the convenience of separating the spaces, it is a more applicable data structure. Nowadays, many games will use quadtree as the tool of collision detection. Also, any objects that are built by composing spaces can use this structure.

ii、 Types of Quadtree

There are many types of quadtrees. The types are classified by the data stored inside the node or the way it partitions the space. I'm going to only talk about the three common types.

01、Region Quadtree

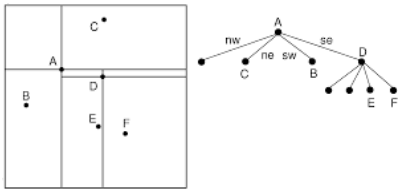
Every node in a region quadtree represent the whole area that this node includes. The data type that store in the nodes can be very different. It can be numbers, colors, words, phrases, etc. So, the region quadtree is also a type of trie that has specific keys to each node. Due to its various data type, region quadtree has many applications, like image, data field representation, temperatures' display of a city.

02、 Point Quadtree

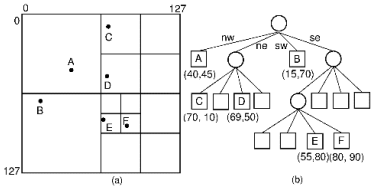
It is the original adaption of binary tree. It stores point data to represent the points' distribution in a 2-D dimensional space. Every quadrant division is centered on the point that inserted into the quadtree is the main characteristic of a point quadtree. When you insert a point into the quadtree, the point will find its corresponding cell and divide the cell into four quadrants with the center of the given points. With this implementation, the subdivided region isn't guarantee as a square.

03、Point Region Quadtree

It is a composition of a region quadtree and a point quadtree. It stores the points in each leaf node but divide the quadrant into four equal spaces. That is, there is no leaf node contains more than one point. If there are more points in one same area, then the division will happen. So, if an area contains zero or one points then it only consists one single leaf node. Or else, there will be four leaf nodes which is used to represent the split spaces.



▲Point Quadtree



▲Point Region Quadtree

II. Comparison

i、 Binary Tree Comparison

I will compare the difference and similarity with the binary tree.

01、Differences

First, quadtree has four children node but binary tree only has two. Second, quadtree partition the 2-D space by the relative position but binary tree sets the nodes by its numerical value comparison. Third, the data type store in each node may differ in a quadtree, some may be the value represent the whole area, some may be the points in the correspond area. The binary tree's data type mostly is the numerical type, which can compare the order instantly.

02、Similarity

First, both of them construct its own tree by comparison and relative relationship. Second, they are all tree structure that has parents belong to their children. Third, when they do the search or insert function in the average case, they only have to choose one way from 2 or 4 total paths to get the result which could reduce the operation time.

ii、 Comparison Chart

	Children	**Partition Space**	**Partition Way**	**Data Type**
Quadtree	4	Two-dimensional	Position relationship	Differ from types
Binary tree	2	One-dimensional	Numerical comparison	Numerical value

▲ Differences Chart

	Structure Type	**Relationship with Parents**	**Search and Insert Method**
Quadtree	tree	Yes (parents are the median)	Through one of the paths O(log n)
Binary tree	tree	Yes (parents are the correspond region)	Through one of the paths O(log n)

▲ Similarity Chart

III. Implementation

In this section, I will talk about the implementation of quadtree by using the region quadtree as an example. Points are only stored in the leaf node. Every parent node will represent the subdivided region. In the following paragraph, I will focus on four must-have functions.

i、 Quadtree Class

Before we start on the functions of quadtree, we need to have a quadtree class first. We also need to build some 2-D dimensional shape class because quadtree is a tool to partition the spaces. Here, I am going to build a point class that has two private variables which are its x and y coordinates so that we can store the points into the leaf node. Second, I will build another class for rectangle which has four private variables that are coordinates, width, and height and one public function to show if a point is inside your shape which is called `contains()`. So,

the basic preparation for a quadtree class is finished. Then, we could go straight into the quadtree class. The quadtree class would have five public functions that I will talk later and four private variables which are capacity, pointset, boundary, and divided. `Capacity` is an integer type and mostly set as four in a quadtree. `Pointset` is an array in points type that stores the points that consist in the quadtree. `Boundary` is a rectangle type and shows that how big the quadrant is. `Divided` is a bool type and tells whether this quad has been divided to smaller spaces or not.

```
class points
{
    private:
          int x, y;
};
class rect
{
    public:
          contains(){}
    private:
          int x, y, w, h;
};
```

▲Point / Rectangle Class

```
class Quadtree()
{
   public:
      Quadtree();
      void insert();
      void subdivide();
      bool search();
      points* range();
   private:
      int capacity;
      points pointset[];
      rect boundary;
      bool divided;
};
```

▲Quadtree Class

```
Quadtree(rect r0, int c)
{
  capacity = c;
  boundary = r0;
  divided = false;

  //children
  Quadtree* TopLeft = null;
  Quadtree* TopRight = null;
  Quadtree* BotLeft = null;
  Quadtree* BotRight = null;
}
```

▲Quadtree Constructor

ii、 Constructor

After we finished the quadtree setting, we start with the constructor to build the first quadtree. We have to input the boundary and capacity into the constructor so that the quad's area can be set. However, there isn't any points or information in the quadtree yet. Then we need to set the divided variable to be `false` and set the children node to be `NULL`. After these steps, the empty quadtree is build and we can start putting points into it.

iii、 Insert ()

We call the insert function when we want to add a new point into the existing quadtree. When we call the insert function, we will have to make sure that we input a point that it is in the area. We will have to do the boundary check to make sure. After the checking process, we will find the correspond quadrant that the point belongs to. The match region finding will process continually until the region isn't divided anymore. Then the points should belong to one of the leaf nodes. However, as we known from the introduction that the leaf node can only have at most one point, the final insert step will have two situations to deal with.

01、Situation 1 — regular insert

We are in the insert function's situation one if the leaf node which we find through the matching process didn't consist any point. We can just push the points into the node in this case. Then our insertion of this point is finished.

02、Situation 2 — need to subdivide into smaller quadrant

We are in the insert function's situation two if the leaf node which we find through the matching process already has one point inside. Then we will have some more processes to do to complete the insert. First, we call the subdivide function to help us divide the correspond quadrant. After the division, we have to go to the children quadtree to see if

the condition is satisfied. We will do the process repeatedly until the leaf node only consists at most one point. When we found respective leaf node for each point then the insertion is done.

iv、 Subdivide ()

We will call this function when the insert is in situation two. We have to divide the region into four equal quadrant and make the original node become a new quadtree. Because I do the region quadtree example here, the subdivide function will have to insert the original point in the node into the new quadtree so that it will still stick to the construction requirements. Then the subdivide process is finished.

v、 Search ()

We call the search function when we want to know if the point is inside our quadtree. Also, you can modify your search function to make it return the closet point to the given one. I am going to only show the example of returning whether the given point is in the quadtree. Same as the insert function, we have to do the boundary check first. Then we will do the region matching process until we find the correspond region that isn't divided yet. Finally, we will match the two point that belong in the same quadrant to see if they are same or not. If they equal to each other, we will return true. Or else, we return false.

vi、 Range ()

Range function has the same concepts of search function. The only difference is that we have to input a shape not a point when we call the function and we return a set of points that is in the range not a bool value. The other processing steps are all same as above. First, we do the boundary check. Second, we find the overlapping region and get the points in that region and use the contains function of that shape to see if the points are in the given region. If it is, we push the points into our returning set. Finally, we return the set of points in the range when we finish checking all the overlapping area.

IV. Examples of Application in Life

Since quadtree is an applicable data structure, there are many applications that use quadtree to complete. I will talk about two famous and common applications of quadtree here.

i、 Collision Detection

Most of the 2-D or 3-D online battle games like shooting or Arena of Valor use quadtree to detect the collision. For example, if the enemy launch an attack with gun or bomb, it will have a shooting or blasting range. Then the display has to react the influence instantly. As we known from above that the search or range function in the quadtree will reduce lots of operation time, many game developers will use point quadtree or point region quadtree to store the position of all the objects and characters. Later, we will call the range function with the attack range and mark those points which are inside the range then make some changes of their status due to the attack. With the shorter operation time, we can find out that the game will have smaller possibility to stuck or stop because of the massive calculation.

ii、 Image Processing

We know that an image is composed of pixels. Different sizes of pixels in the same image represent different quality. With smaller but more pixels, we have better quality. Also, an image is a 2-D dimensional space. Then, with the characteristic of the 2-D dimensional space, we can use the region quadtree to represent the picture. Every node in the quadtree will store the RGB value of that quadrant. With another characteristic that each picture will have same size of pixels if we decide to have a specific quality, the quadtree will be a complete tree. It's because that we have to subdivide each region into same size. That is, if we have N nodes in the last level, then the upper level will have N/4 nodes. The numbers of all the nodes in the quadtree will be a geometric series with the common ration of 1/4. Each level of nodes will represent correspond quality of the picture. Moreover, in image processing, there is a big difference from the above implementation. We will construct an image quadtree from its leaf node and get its parent node from getting the average RGB values of the children node. So, the insertion time complexity here is O(n). The benefit to use quadtree to build an image is that it can reduce the memory storage size. One way is that we can store only one level of pixel color but we can get the above information by some calculation. The other way is that we can do some "pruning" action to our quadtree. It means that we will remove the leaf that has similar colors with its parents. In this case the numbers of nodes are reduce, but the quality is not changed.

V. Analysis

i、 Pros

First, the quadtree is famous for its efficiency in the search and range operation. With the above explanations of search and range function, we can know that the average time complexity reduces a lot. With little operation time, we can use it in more games so that the gaming experiences will be better. Our characters and shooting display won't stuck or black out. Second, in the application aspect, if we implement a quadtree into the image processing, we can use the "pruning" technic which can compress the image but without reducing the quality. In this way we can reduce a lot of storage spaces. Third, continue on the image processing, with different levels of quadtree node, we can easily get different resolution of the picture. It can help us vary the resolution quickly.

ii、 Cons

First, the quadtree will have a good performance only when your information is random enough, or else the time complexity is still as bad as before. That's because if two point are very close, you will need to partition the spaces into very small so that two points will belong to different quadrant. In this kind of case, the tree will become very unbalanced. The time complexity then will turn to the worst case not the average case we preferred. Second, the quadtree has four children and every child has another four children. When we have a big area with many objects, we will have a massive number of pointers. The spaces we use will be very many if we want to build a large space. It will be an expensive cost. Third, quadtree is built by the relative position

so if an object is moved or the picture is rotated then we have to rebuild the whole tree. Every branch or the times of divisions may all be different if we have all these adjustments. Then, there will be another expensive cost.

iii、 Time Complexity

These functions will have a better time complexity of $\log n$ is because we only have to go through one of the paths to get the result when we do the insertion or search process. Then, the operation time will become a lot less if we compare with other functions that do the process through all the objects. I organize a chart below for four important functions' time complexity. Those complexity were calculated based on the implementations I did in the previous section.

	Worst Case	Average Case
Insert ()	O(n)	O(log n)
Subdivide ()	O(1)	O(1)
Search ()	O(n)	O(log n)
Range ()	O(n)	O(log n)

▲ Time Complexity Chart (n: the numbers of nodes)

VI. Conclusion

I want to know some data structures that not only store the numerical value but also other data types like points, areas, colors, etc. Also, I'm interesting about how we can store these different types of data in an efficient way. Moreover, I wonder is there any data structure that are extended from the class and often applied in our life. Quadtree became my topic because it satisfies all my three questions. After I dig deeper into quadtree, I found out more interesting and useful things about it. The collision detection and image processing are the two things that impressed me a lot. Last, I would say quadtree is a good and bad data structure. It has its benefit when the information is random and with appropriate size. Therefore, if these conditions aren't not made, quadtree become useless. That is, when we want to implement a quadtree, we have to be careful of its data. In this way, we can maximize the value of quadtree.

VII. References

https://en.wikipedia.org/wiki/Quadtree

https://www.geeksforgeeks.org/quad-tree/

https://www.geeksforgeeks.org/image-manipulation-using-quadtrees/

https://davidhsu666.com/archives/quadtree_in_2d/

https://www.youtube.com/watch?v=OJxEcs0w_kE

https://gamedevelopment.tutsplus.com/tutorials/quick-tip-use-quadtrees-to-detect-likely-collisions-in-2d-space--gamedev-374

http://euklid.mi.uni-koeln.de/c/mirror/www.cs.curtin.edu.au/units/cg351-551/notes/lect5e1.html

http://blog.roy4801.tw/2020/07/19/ds/quadtree/

https://www.cs.cmu.edu/~ckingsf/bioinfo-lectures/quadtrees.pdf