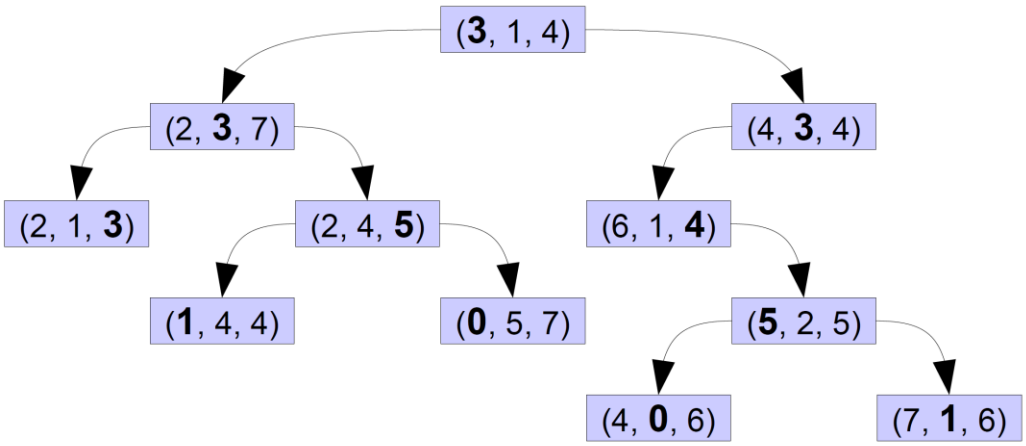
# Lab5: 2D-Tree

In this assignment, you will implement a special data structure called a 2D-Tree (short for “2-dimensional tree”).

## 1. Story

At a high level, a kd-tree(short for “k-dimensional tree”) is a generalization of a binary search tree that stores points in k-dimensional space. That is, you could use a kd-tree to store a collection of points in the Cartesian plane, in three-dimensional space, etc. You could also use a kd-tree to store biometric data, for example, by representing the data as an ordered tuple, perhaps (height, weight, blood pressure, cholesterol). However, a kd-tree cannot be used to store collections of other data types, such as strings. Also note that while it's possible to build a kd-tree to hold data of any dimension, all of the data stored in a kd-tree must have the same dimension. That is, you can't store points in two-dimensional space in the same kd-tree as points in four-dimensional space.

It's easiest to understand how a kd-tree works by seeing an example. Below is a kd-tree that stores points in three-dimensional space:




Notice that in each level of the kd-tree, a certain component of each node has been bolded. If we zero-index the components (i.e. the first component is component zero, the second component is component one, etc.), in level n of the tree, the (n % 3)-th component of each node is shown in bold. The reason that these values are bolded is because each node acts like a binary search tree node that discriminates only along the bolded component. For example, the first component of every node in the left subtree is less than the first component of the root of the tree, while the first component of every node in the right subtree has a first component at least as a large as the root node's. Similarly, consider the kd-tree's left subtree. The root of this tree has the value (2, 3, 7), with the three in bold. If you look at all the nodes in its left subtree, you'll notice that the second component has a value strictly less than three. Similarly, in the right subtree the second component of each node is at least three. This trend continues throughout the tree.

Given how kd-trees store their data, we can efficiently query whether a given point is stored in a kd-tree as follows. Given a point P, start at the root of the tree. If the root node is P, return the root node. If the

first component of P is strictly less than the first component of the root node, then look for P in the left subtree, this time comparing the second component of P. Otherwise, then the first component of P is at least as large as the first component of the root node, and we descend into the right subtree and next time compare the second component of P. We continue this process, cycling through which component is considered at each step, until we fall off the tree or find the node in question. Inserting into a kd-tree is similarly analogous to inserting into a regular BST, except that each level only considers one part of the point.

# 2. Lab Requirement

You should implement the TreeNode and BinaryDimonTree(2D-Tree) classes.

TreeNode is a class which is used to store the binary dimension data. You should provide the following functions (at least):

1. **int getX()**: return x(the first value in the tree node);
2. **int getY()**: return y(the second value in the tree node);

BinaryDimonTree is a class for the Tree which supports the following functions (at least):

1. **void insert(int x, int y)**: insert the Point(x, y) into the tree according to the 2D-Tree rules;
   In this lab, if the Point(x, y) equals to one of the node(N) in the tree, insert the Point(x,y) into the node's(N) **right** subtree.
2. **TreeNode* find_nearest_node(int x, int y)**: search the 2D-Tree and find the point whose distance is smallest to the Point(x, y), and output the result to the console;
   p.s. the distance between Point(x1, y1) and Point(x2, y2) is equal to [(x1-x2)^2 + (y1-y2)^2]^(1/2);
3. **~BinaryDimonTree()**: the deconstruct function which is used to reclaim all the data in the 2D-Tree, and you should explain your implementation to us when your lab is examined.

You must implement the Tree completely by yourself. And **the names of the classes and their functions must be same with the above**.

We have provided the main function and the test cases, you should just run the main.cpp and check if you can pass through all the test cases.

# 3. Guidance

The algorithm to find the nearest node in the 2D-Tree:

```
Let the test point be (a0, a1).
Maintain a global best estimate of the nearest neighbor, called 'guess.'
Maintain a global value of the distance to that neighbor, called 'bestDist'
Set 'guess' to NULL.
Set 'bestDist' to infinity.
Starting at the root, execute the following procedure:
    if curr == NULL
        return
```

```
/*
 * If the current location is better than the best known location,
 * update the best known location.
 */
if distance(curr, guess) < bestDist
    bestDist = distance(curr, guess)
    guess = curr

/*
 * Recursively search the half of the tree that contains the test point.
 * i means the dimension index which is used to discriminate in the current
   level,
 * for example:
 * the i = 0 when curr is the root(level 0),
 * the i = 1 when curr is the node in level 1;
 * the i = 0 when curr is the node in level 2;
 * and so on.
 */
if a_i < curr_i
    recursively search the left subtree on the next axis
else
    recursively search the right subtree on the next axis

/*
 * If the vertical distance between the points is smaller than the best
   distance,
 * look on the other side of the plane by examining the other subtree.
 */
if |curr_i - a_i| < bestDist
    recursively search the other subtree on the next axis
```

if you have some problems understanding the algorithm, you can find more explanation in the **kdtree.pdf** in the folder of this lab.

# 4. Upload

提交时，请将
- 你完成的源代码压缩成 zip 压缩包，并重命名为`lab5-XXX.zip`;
上传到: [ftp://dmkaplony:public@public.sjtu.edu.cn:/upload/c++2019/lab5/]中。
（其中 XXX 为学号，如`lab5-518037910001.zip`和`lab5-518037910001.pdf`）

如果需要更改，请在文件名后加版本号，最终以最高版本号为准。如第二次提交可用`lab5-518037910001-2.zip`。