**CS2040S: Data Structures and Algorithms**

Discussion Group Problems for Week 7

*For: March 3–March 8*

# Check in

Discuss questions, if you have any, with the tutor and the rest of the class, about the material and content so far.

# Problems

**Problem 1. AVL vs Trie**

Discuss the trade-offs of using AVL and Trie to store strings.

**Ans:** Using an AVL tree to store strings would result in a runtime of O for both inserting and searching whereas using a Trie to store strings would have a runtime of O for inserting and searching, where *L* is the length of a word.

Space wise both the AVL tree and Trie would take up O(total\_string\_length) but Trie would tend to have more overhead cost.

1 node per string for AVL tree vs 1 node per character for Trie as well as links, Trie has more links as compared to AVL trees

**Problem 2. kd-Trees**

A kd-tree is another simple way to store geometric data in a tree. Let’s think about 2dimensional data points, i.e., points (*x,y*) in the plane. The basic idea behind a kd-tree is that each node represents a rectangle of the plane. A node has two children which divide the rectangle into two pieces, either vertically or horizontally.

For example, some node *v* in the tree may split the space vertically around the line *x* = 10: all the points with *x*-coordinates ≤ 10 go to the left child, and all the points with *x*-coordinates *>* 10 go to the right child.

Typically, a kd-tree will alternate splitting the space horizontally and vertically. For example, nodes at even levels split the space vertically and nodes at odd levels split the space horizontally. This helps to ensure that the data is well divided, no matter which dimension is more important.

There are 2 main ways to implement kd-tree.

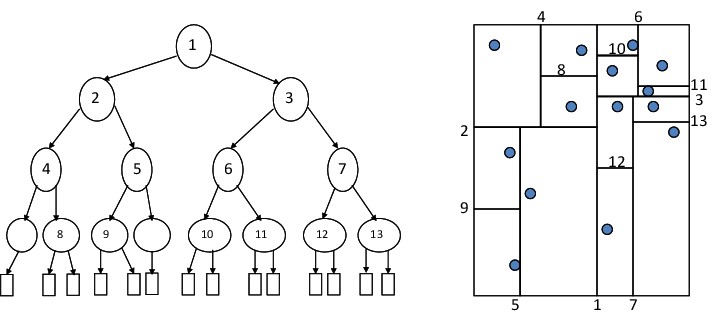
In one way, internal nodes of the tree corresponds to the horizontal or vertical splitting lines; When you have a region with only one data point, instead of dividing further, simply create a leaf which corresponds to the only data point. Figure 1 is an example of a kd-tree that contains 14 points (internal nodes without any label can be omitted):

Another way to implement kd-tree uses horizontal or vertical lines that pass through data points for splitting. In this case, each tree node (internal and external) represents one data point. Figure 2 is an example of a kd-tree that contains 10 points:

**Problem 2.a.** How do you search for a point in a kd-tree? What is the running time?

**Ans:** Firstly, start at the root. At each node, there is a horizontal or vertical split. If the node has a vertical split, compare the -coordinate. If the -coordinate of the point is smaller than the node’s -coordinate, branch left, else we branch right. The following node would then be a vertical split where the process repeats. The running time for the search operation is O(h) where h is the height of the kd-tree.

**Problem 2.b.** You are given an (unordered) array of points. What would be a good way to



**Figure 1:** On the left: the points in the input. On the right: how the points are stored in the kd-tree

build a kd-tree? Think about what would keep the tree nicely balanced. What is the running time of the construction algorithm?

**Ans:** At each level, depending if we need to split the tree horizontally (-coordinate) or vertically -coordinate), we can use QuickSelect to identify the median of the or -coordinate. We can then recursively construct the remaining tree on the two subarrays partitioned around the median. This partitioning step is O and the tree would have a height of . The total cost of the construction algorithm is .

**Problem 2.c.** How would you find the element with the minimum (or maximum) x-coordinate in a kd-tree? How expensive can it be, if the tree is perfectly balanced?

**Ans:** If we wish to find the minimum -coordinate, suppose the node is a horizontal split, we simply recurse on the left subtree. Since the next node is a vertical split, the minimum -coordinate may be in either subtrees. Thus, we would need to recurse on both subtrees. The recurrence relation for this operation is . At each step down the tree, the number of points divides in half, i.e. points after one step and after another step. There will be two recursive calls being made after every alternate step. The tree that is recursed on would have a height of , and at each node, O(1) work is being done to check whether to recurse on the left or right. A tree of height would have nodes. Thus, the total running time is O.

**Problem 3. Tries(a.k.a Radix Trees)**

Coming up with a good name for your baby is hard. You don’t want it to be too popular. You don’t want it to be too rare. You don’t want it to be too old. You don’t want it to be too weird.[[1]](#footnote-1)

Imagine you want to build a data structure to help answer these types of questions. Your data structure should support the following operations:

insert(name, gender, count): adds a name of a given gender, with a count of how many babies have that name.

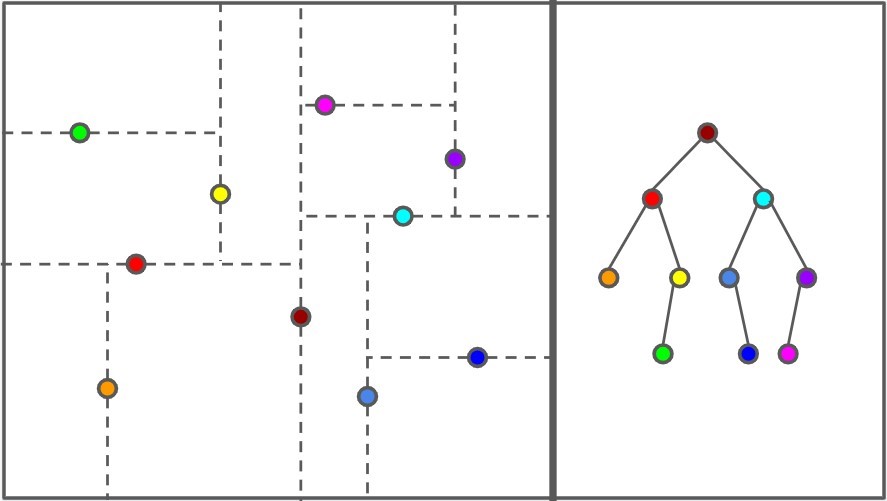
countName(name, gender): returns the number of babies with that name and gender.

countPrefix(prefix, gender): returns the number of babies with that prefix of their name and gender.

countBetween(begin, end, gender): returns the number of babies with names that are lexicographically after begin and before end that have the proper gender.

In queries, the gender can be either boy, girl, or either. Ideally, the time for countPrefix should not depend on the number of names that have that prefix, but instead run in time only dependent on the length of the longest name.

**Ans:** We can make use of a Trie to store the names of the babies and the count of names at the end of the node, for each gender.



**Figure 2:** On the left: the points in the input. On the right: how the points are stored in the kd-tree

**Problem 4. A Trie Question?**

**Problem 4.a.** Your task is simple. Given an array of 32 bits unsigned positive integers, find 2 numbers such that their XOR is maximum. (Convert all the bits to 1)

Hint 1: Look at the title of the question.

Hint 2: Think of numbers bit by bit from the most significant bit.

**Problem 4.b.** (Bonus, optional, difficult) (Source: https://codeforces.com/contest/1777/problem/F) Now, we modify the question. You are given an array of 32 bits unsigned positive integers *A*. Find a subarray of A(l,r) such that the following value is maximised:

*Al* ⊕ *Al*+1 ··· ⊕ *Ar* ⊕ *max*(*Al,Al*+1*,...Ar*)

1. The website <https://www.babynamewizard.com/voyager> let’s you explore the history of baby name popularity! [↑](#footnote-ref-1)