CMSC 420: Fall 2022

**Homework 3: Geometric Search and Hashing**

Handed out Tue, Nov 8. Due **Tue, Nov 15, 9:30am** (that is, by the start of class).

**Important!** Solutions will be discussed in class right after the due date, so **no late submissions will be accepted**. Turn in whatever you have completed by the due date.

**Problem 1.** (10 points) Consider the kd-tree shown in Fig. 1. Assume that the cutting dimensions alternate between $x$ and $y$.

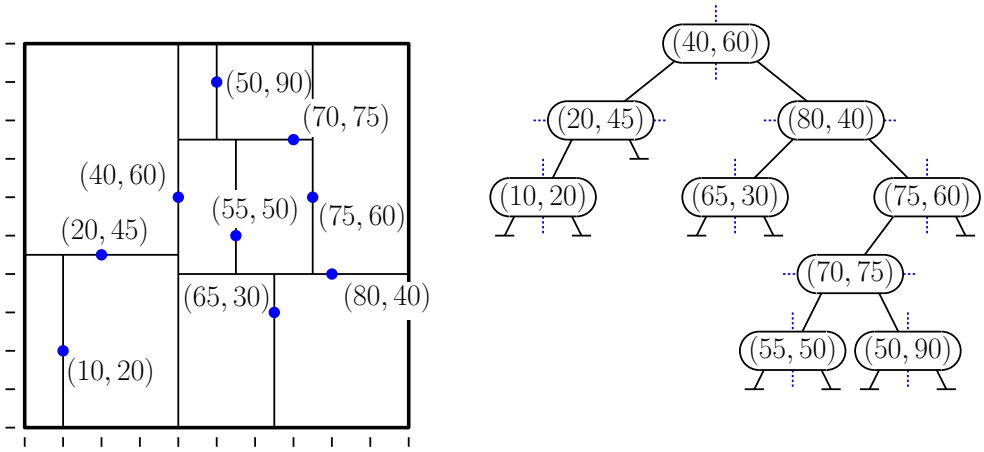


Figure 1: kd-tree operations.

(a) (5 points) Show the result of inserting $(30, 30)$ into this tree. As in Fig. 1 show both the tree structure (with cutting directions indicated) and the subdivision of space.

(b) (5 points) Show the kd-tree that results after deleting the point $(40, 60)$ from the *original* tree. Indicate which node is the replacement node, and show both the tree structure and the subdivision of space.

(Intermediate results are not required, but may be given to help assigning partial credit.)

**Problem 2.** (5 points) Suppose that we are given a set $P = \{p_1, \ldots, p_n\}$ of $n$ points in $\mathbb{R}^2$ stored in a kd-tree. Recall that the *cell* of a node in the tree is the rectangular region of space associated with this node. Each `null` pointer of the tree is associated with a cell that contains no point of $P$, which we call a *null-pointer cell*. (In Fig. 2, the `null` pointers are shown as small squares.)

We assume that the kd-tree satisfies the *standard assumptions*: (1) The cutting dimensions alternate between $x$ and $y$ with each level of the tree, (2) subtrees are balanced in the sense that if the subtree rooted at a node $p$ contains $m$ points, then its two subtrees each contain roughly $m/2$ points. The following lemma follows from the analysis of orthogonal range searching from class (Lecture 14).

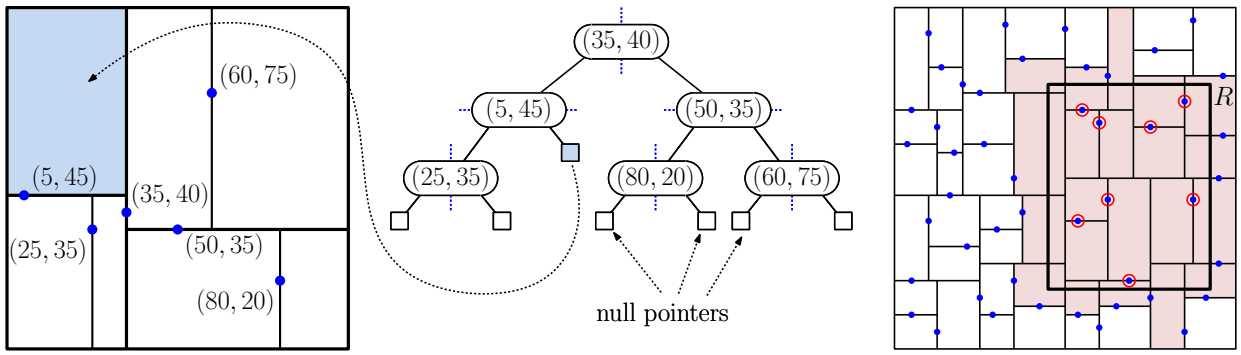


Figure 2: (a) A kd-tree and null-pointer cells and (b) illustration of Lemma B.

**Lemma A:** Given any kd-tree storing a set of $n$ points in $\mathbb{R}^2$ that satisfies the standard assumptions and given any axis parallel line $\ell$, the number of null-pointer cells that intersect $\ell$ is $O(\sqrt{n})$.

In this problem, we will generalize this result. Consider any axis-parallel rectangle $R$, and let $k$ denote the number of points of $P$ that lie within $R$. Prove the following result.

**Lemma B:** Given any kd-tree storing a set of $n$ points in $\mathbb{R}^2$ that satisfies the standard assumptions and given any axis parallel rectangle $R$ that contains $k$ points of the set, the number of null-pointer cells that intersect $R$ is $O(k + \sqrt{n})$.

Note that if a cell is completely contained within $R$, we consider to intersect $R$. For example, in Fig. 2(b), rectangle $R$ contains 8 points of $P$ (circled in red) and it intersects 21 null-pointer cells (shaded in pink).

**Hint:** There is no need to resort to solving recurrences. Classify the null-pointer nodes into two groups, those that are completely contained within $R$ and those that partially overlap $R$.

**Problem 3.** (10 points) As in the previous problem, suppose that we are given a set $P = \{p_1, \ldots, p_n\}$ of $n$ points in 2D space stored in a point kd-tree (see Fig. 3(a)), which satisfies the standard assumptions. Each node stores a point `p.point`, a cutting dimension `p.cutDim`, and left and right child pointers `p.left` and `p.right`, respectively. You may make use of any primitive operations on points and rectangles. You may assume that there are no duplicate coordinate values among the points of $P$ or the query point.

In a *segment sliding*, you are given a vertical line segment $s$, and the query returns the first point $p \in P$ that is hit if we were to slide the segment to its left (see Fig. 3(b)). If a point $p_i$ lies on the segment, then the answer is $p_i$. If there is no point of $P$ to the left of the segment, the query returns `null`. Since there are no duplicate $x$-coordinates, so the answer is always unique.

To simplify argument lists, let's assume that we have access to a class `VertSeg` that stores a vertical segment. Given an object `seg` is of this type, `seg.x` is its $x$-coordinate, `seg.ylo` is its lower $y$-coordinate, and `seg.yhi` is its upper $y$-coordinate.

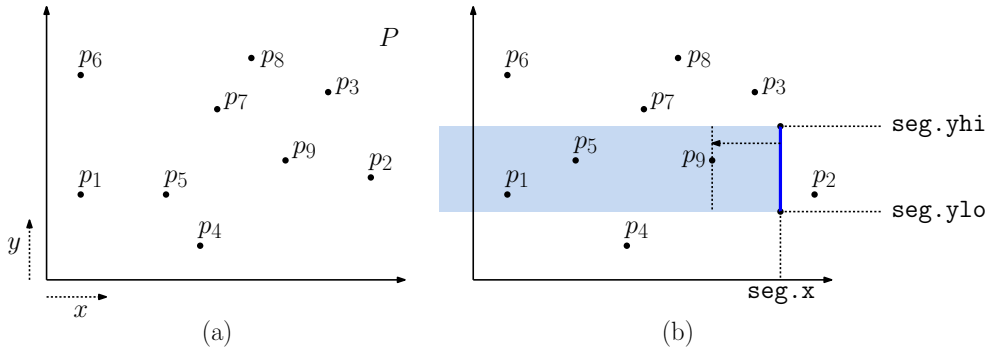


Figure 3: Vertical segment sliding queries.

(a) (7 points) Present pseudo-code for an efficient algorithm, `Point slideLeft(VertSeg seg)` for answering this query given the kd-tree. (**Hint:** As usual, create a recursive helper function and explain how it is initially invoked. This is similar to nearest-neighbor searching in the sense that you will keep track of each node's `cell` and the `best` point seen so far in the search. For efficiency, you should avoid visiting nodes that cannot possibly contain the answer to the query.

(b) (3 points) Prove that your algorithm runs in $O(\sqrt{n})$ time. (**Hint:** Lemma B above is helpful here. It only bounds the number of null-pointer nodes, but you may assume that the bound applies to all the nodes that overlap the rectangle $R$. What is $R$ in this case?)

**Problem 4.** (10 points) In this problem we will consider how to design a data structure for a particular application. You are given a set $P = \{p_1, \ldots, p_n\}$ of $n$ points in $\mathbb{R}^2$. Each point represents the coordinates (e.g., longitude and latitude) of a gas station (or if you prefer, a charging station for your EV). In addition to its coordinates, each point $p_i = (x_i, y_i)$, is associated with a *cost*, denoted $c_i$ (see Fig. 4(a)). Think of this as the cost of refueling or recharging. These points and the costs are fixed, and you may build a data structure for answering the following queries.

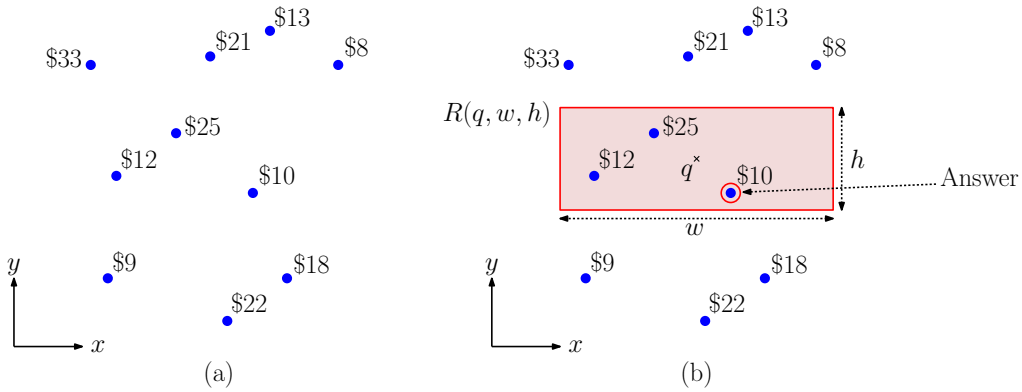


Figure 4: Cheapest fuel.

A query asks what is the lowest cost station in a given rectangular area (say, the map display

on the car's navigation system). Your app knows the user's location, call it $q = (x_q, y_q)$, and there is a rectangle $R(q, w, h)$, called the *query rectangle* of width $w$ and height $h$ centered about $q$ (see Fig. 1). Your query is given $q$, $w$, and $h$, and the answer to the query is the identity of the station in $P$ that lies within $R(q, w, h)$ (see Fig. 4(b)). For simplicity, you may assume that the costs are all distinct, so the answer is always unique.

Present an efficient data structure and algorithm for answering these queries. Our objective is a query time of $O(\log^a n)$ using space $O(n \log^b n)$, where $a, b > 0$ are small constants. (**Hint:** Explain how to modify range trees. Note that you cannot use a kd-tree, since that would have a query time of at least $O(\sqrt{n})$. Also note that you do not have time to list all the points inside the query rectangle, since that number could be as high as $n$.)

(a) (5 points) Describe your data structure, and derive its space bound. (**Hint:** Use a variant of 2-D range trees. What cost-related information is stored in each node of the tree?)

(b) (5 points) Explain how queries are answered, and derive the query time. (Note: Pseudocode is not required. A high-level English description is fine.)

**Problem 5.** (15 points) In this problem, you will show the result of inserting a sequence of three keys into a hash table, using linear and quadratic probing and double hashing. In each case, indicate the following:

- Was the insertion successful? (The insertion fails if the probe sequence loops infinitely without finding an empty slot.)
- If the insertion is successful, indicate the number of *probes*, that is, the number of array elements accessed. (The final location you place the key counts as a probe, so the number of probes is one more than the number of collisions encountered.)
- Show contents of the hash table after each insertion. (You will show three tables for each part.)

For the purposes of assigning partial credit, you can illustrate the probes made as we did in the lecture notes (with little arrows).

(a) (5 points) Show the results of inserting the keys "`X`" then "`Y`" then "`Z`" into the hash table shown in Fig. 5(a), assuming *linear probing*. (*Insert the keys in sequence, so if all are successful, the final table will contain all three keys.*)

**(a) Linear probing**

```
insert("X")  h("X") = 4
insert("Y")  h("Y") = 6
insert("Z")  h("Z") = 0
```

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| G | | F | | E | A | D | | C | B |

**(b) Quadratic probing**

```
insert("X")  h("X") = 6
insert("Y")  h("Y") = 3
insert("Z")  h("Z") = 8
```

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| | | | B | D | | A | | E | C |

**(c) Double hashing**

```
insert("X")  h("X") = 14; g("X") = 6
insert("Y")  h("Y") = 5; g("Y") = 3
insert("Z")  h("Z") = 4; g("Z") = 4
```

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| E | | C | | F | A | | | G | | D | | H | | B |

Figure 5: Hashing with linear probing, quadratic probing, and double hashing.

(b) (5 points) Repeat (a) using the hash table shown in Fig. 5(b) assuming *quadratic probing*. (**Hint:** You may wish to use the fact that for any integer $i \geq 0$, the value of $i^2$ mod 10, is one of $\{0, 1, 4, 5, 6, 9\}$.)

(c) (5 points) Repeat (a) using the hash table shown in Fig. 5(c) assuming *double hashing*, where the second hash function $g$ is shown in the figure.

**Challenge Problem:** In Problem 2 above we analyze the number of null-pointer cells that intersect any axis-parallel line in a kd-tree in $\mathbb{R}^2$ with $n$ points. In this problem, let's consider two natural generalizations to $\mathbb{R}^3$. Again, we have $n$ points, and let us make the *standard assumptions* that the cutting dimensions cycle among $x$, $y$, and $z$, and subtree sizes are balanced.

(a) A plane is *axis-orthogonal* if it is orthogonal to one of the three coordinate axes. For example, the set $\{(x, y, z) \mid y = 13.4\}$ is a plane orthogonal to the $y$-axis that cuts the $y$-axis at 13.4.

Show that the number of null-pointer cells in a stanard-assumption kd-tree that intersect any axis-orthogonal plane is $O(n^c)$ for some constant $0 < c < 1$. Derive the value of $c$.

**Hint:** Derive a simple recurrence and apply the *Master Theorem* to solve it.

(b) An (infinite) line is *axis-parallel* if it is parallel to one of the three coordinate axes. For example, the set $\{(x, y, z) \mid x = 4, z = 7\}$ is a line parallel to the $y$-axis that pierces the $(x, z)$-coordinate plane at the point $(4, 0, 7)$.

Repeat part (a) for any axis-parallel line.