Problem Set 6

MIT students: This problem set is due in lecture on *Monday, October 29*.

SMA students: This problem set is due after the video-conferencing session on *Wednesday, October 31*.

Reading: Chapter 14; Chapter 33 (§33.1–33.3)

Both exercises and problems should be solved, but *only the problems* should be turned in. Exercises are intended to help you master the course material. Even though you should not turn in the exercise solutions, you are responsible for material covered by the exercises.

Mark the top of each sheet with your name, the course number, the problem number, your recitation instructor and time, the date, and the names of any students with whom you collaborated.

MIT students: Each problem should be done on a separate sheet (or sheets) of three-hole punched paper.

SMA students: Each problem should be done on a separate sheet (or sheets) of two-hole punched paper.

You will often be called upon to "give an algorithm" to solve a certain problem. Your write-up should take the form of a short essay. A topic paragraph should summarize the problem you are solving and what your results are. The body of your essay should provide the following:

1. A description of the algorithm in English and, if helpful, pseudocode.
2. At least one worked example or diagram to show more precisely how your algorithm works.
3. A proof (or indication) of the correctness of the algorithm.
4. An analysis of the running time of the algorithm.

Remember, your goal is to communicate. Graders will be instructed to take off points for convoluted and obtuse descriptions.

Exercise 6-1. Do exercise 14.1-5 on page 307 of CLRS.

Exercise 6-2. Do exercise 14.2-2 on page 310 of CLRS.

Exercise 6-3. Do exercise 14.3-1 on page 316 of CLRS.

Exercise 6-4. Do exercise 14.3-5 on page 317 of CLRS.

Exercise 6-5. Do exercise 33.1-4 on page 946 of CLRS.

Exercise 6-6. Do exercise 33.2-1 on page 946 of CLRS.

Problem 6-1. Overlapping rectangles

VLSI databases commonly represent an integrated circuit as a collection of rectangles. Assume that each rectangle is rectilinearly oriented (sides parallel to the x- and y-axis), so that a representation of a rectangle consists of its minimum and maximum x- and y-coordinates.

(a) Give an $O(n \lg n)$-time algorithm that decides whether a set of rectangles so represented contains two rectangles that overlap. Your algorithm need not report all intersecting pairs, but it must report that an overlap exists if one rectangle entirely covers another, even if the boundary lines do not intersect. (*Hint:* Move a "sweep" line across the set of rectangles by replacing one of the two spatial dimensions with time. At all times maintain the collection of rectangles pierced by the sweep line.)

(b) Argue that your algorithm indeed runs in $O(n \lg n)$ time in the worst case.

Problem 6-2. GPS map display

After graduating from MIT, you decide to join GiPSy, a startup that hopes to make money by selling an affordable hand-held GPS (Global Positioning System) device. The device picks up time-stamped messages from 4 geostationary satellites and uses them to calculate its precise coordinates on the globe by taking into account the orbital position of each satellite (included in the messages) and the time it took each message to reach the device.

The device has a rectangular LCD screen which displays the user's exact location on the map, as well as all the nearby streets; see Figure 1. The device must be able to update the map display in real time because the user may be moving and may zoom in or out. GiPSy's first model will only work in cities whose streets are arranged in a grid (such as Manhattan). If the first model proves to be a success in those markets, GiPSy hopes to secure the necessary funding to develop a model that can work in all cities.

Your task is to figure out how the city map should be preprocessed and stored on the device, in order to quickly answer queries about what streets are in the vicinity of the user. More precisely, the problem is defined as follows:

- A ***road*** $r = \langle (r_{x1}, r_{y1}), (r_{x2}, r_{y2}) \rangle$ is a line segment, either horizontal or vertical, specified by the coordinates of its endpoints, (r_{x1}, r_{y1}) and (r_{x2}, r_{y2}). Thus, for every road $r = \langle (r_{x1}, r_{y1}), (r_{x2}, r_{y2}) \rangle$, we have either $r_{x1} = r_{x2}$ (vertical) or $r_{y1} = r_{y2}$ (horizontal).

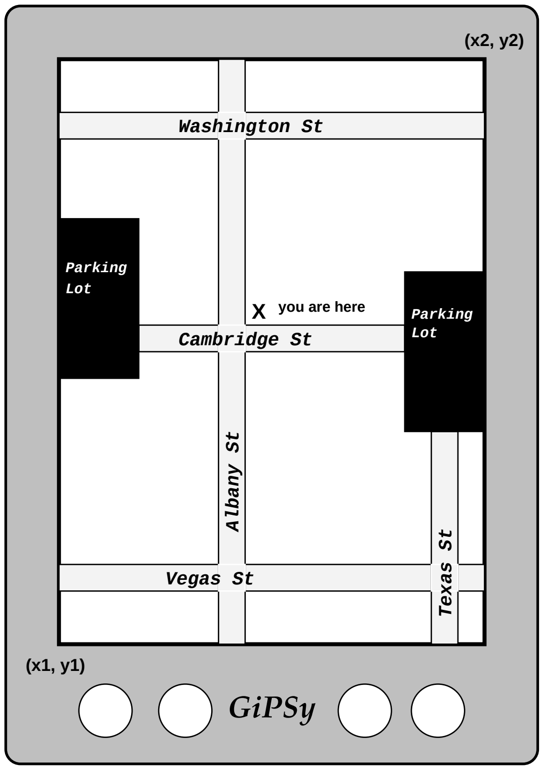

Figure 1: A sketch of a GiPSy device. Washington St., Vegas St., and Albany St. are Type-2 roads. Cambridge St. and Texas St. are Type-1 roads. The view rectangle is determined by the coordinates of the lower-left and upper-right corners, as shown.

- A ***map*** $M = \{r^1, r^2, \ldots, r^n\}$ is a set of n roads.
- A ***view rectangle*** $V = \langle (V_{x1}, V_{y1}), (V_{x2}, V_{y2}) \rangle$ specifies the rectangular region that should be displayed by giving the coordinates of the rectangle's lower-left and upper-right corners, (V_{x1}, V_{y1}) and (V_{x2}, V_{y2}) respectively.
- A road r is ***visible*** in the view rectangle V if it intersects the interior of the rectangle V. There are two types of visible roads r:

  Type 1: One or both endpoints of the road r are inside the view rectangle V.

  Type 2: The road r crosses the view rectangle V but both its endpoints lie outside the view rectangle V.
- The goal of a ***clipping query*** is to report all the visible roads for a given view rectangle V.

Because the map M does not change often, we are free to spend a reasonable amount of time preprocessing the map M into a data structure that supports queries efficiently, using a reasonable amount of auxiliary space.

In the problem parts that follows, you will often be called upon to "give an efficient method" for supporting a particular type of query. For each such problem part, you must do the following:

1. Give an efficient algorithm for preprocessing the map M into a data structure.
2. Give an efficient algorithm for using this data structure to answer the query for a given view rectangle V.
3. Analyze the worst-case preprocessing time, worst-case query time, and worst-case space occupied by the data structure. In all cases, the analysis should be in terms of the total number n of roads on the map M and the number k of visible roads in the view rectangle V.

Optimizing the query time is most important; the preprocessing time is secondary.

(a) Give an efficient method for finding all Type-1 roads. (*Hint:* Use a two-dimensional range tree.)

The rest of the problem is about finding Type-2 roads. Without loss of generality, we focus on finding horizontal Type-2 roads.

A horizontal road r ***straddles*** a view rectangle V if it crosses the left edge of V. Thus, every straddling road r is visible. Depending on whether the right endpoint of a straddling horizontal road r is inside the view rectangle, a straddling horizontal road r may be Type-1 or Type-2.

(b) Suppose that we knew how to compute the set of straddling horizontal roads for a given map M and view rectangle V. Give an efficient algorithm to convert the set of straddling horizontal roads into the set of Type-2 horizontal roads.

Thus, our goal is to identify which horizontal roads are straddling. We use the following characterization. A horizontal road $r = \langle (r_{x1}, r_y), (r_{x2}, r_y) \rangle$ straddles the view rectangle $V = \langle (V_{x1}, V_{y1}), (V_{x2}, V_{y2}) \rangle$ if it satisfies two properties:

1. $r_{x1} \le V_{x1} \le r_{x2}$, i.e., the road r crosses the vertical line extending the left edge of the view rectangle V. We say that r is ***horizontally straddling***.
2. $V_{y1} \le r_y \le V_{y2}$, i.e., the road r falls within the vertical extent of the view rectangle V. We say that r is ***vertically straddling***.

(c) Draw a picture showing a view rectangle V and an example of each of the following kinds of horizontal roads:

1. straddling and Type-2,
2. straddling but not Type-2,
3. horizontally straddling but not vertically straddling,
4. vertically straddling but not horizontally straddling, and
5. neither horizontally straddling nor vertically straddling.

A simple approach for computing the set of straddling horizontal roads is to

- compute the set of vertically straddling horizontal roads, and
- remove from this set any roads that are not horizontally straddling.

(d) Give an efficient method for finding all vertically straddling horizontal roads. (*Hint:* Use a one-dimensional range tree.)

Once we have the set of vertically straddling horizontal roads, we can simply run through the list and remove any roads that are not horizontally straddling. This filtering results in the set of straddling horizontal roads.

(e) Analyze the worst-case running time of this filtering algorithm.

(f) Explain why this approach is slow in the worst case.

Our final goal is to develop a faster method for computing the set of straddling horizontal roads.

(g) Show that instead of explicitly enumerating all vertically straddling horizontal roads in part (d), we can find $O(\lg n)$ nodes in the range tree whose descendants contain all of the vertically straddling horizontal roads and no other roads. Give an efficient algorithm to find these $O(\lg n)$ nodes.

(h) Give an efficient method for finding all the straddling horizontal roads. (*Hint:* Combine interval trees with range trees.)

(i) Conclude by analyzing the entire method for answering clipping queries.