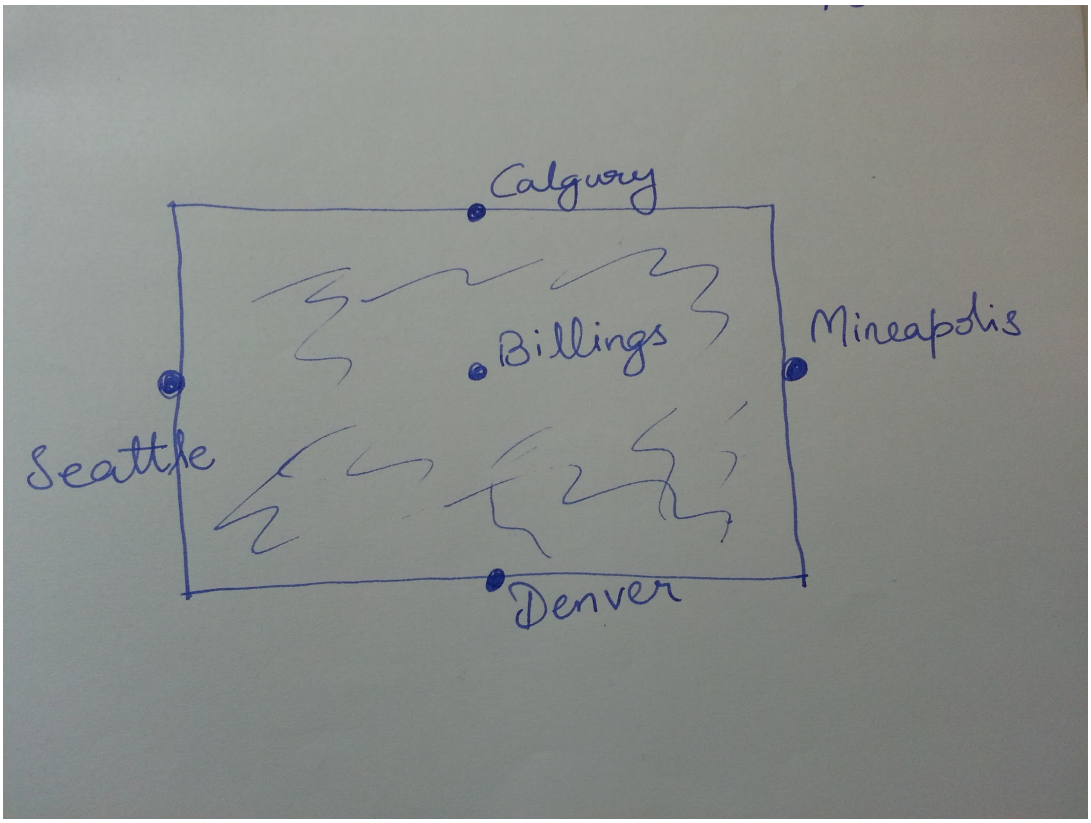

Figure 1: Problem

Algorithm Seminar
September 21, 2012

Problem

Billings, Montana is a special city. Special because it is largest city which is north of Denver, east of Seattle, south of Calgury, and west of Mineapolis. This forms a rectangle as shown in the figure **??**.

This special feature leads to a problem, which can be stated as follows:
Given n cities, their location and the population. Now if you need to start a new city with a given population and a location, what would be the rectangle in which your city dominates (i.e. the largest rectangle enclosing that city, with no other city in the rectangle having population greater than your city's population).

Background

This problem came in picture in 1960s when they wanted to find empty areas on the circuit boards. Algorithms for this started coming in 1970s. Till date more efficient algorithms keep getting published.

Simplified problem

Consider the problem in 1D. You have n cities on a line. Each of the cities has some population. Your city will be one of the points on this line.Given the population of your city, you need to find the segment on the line, which your city will dominate.

Solving the problem in 1-Dimension

The problem can be approached in 2 ways:

1. Online
2. Offline

Offline

1. Given the population of size N. Delete all cities having population less than N.
2. If the cities are sorted in the order of their position, then traverse from left to right and find the left and right city.
3. If the cities are not sorted, this will involve another $nlog(n)$ operations.

Online

Requires a fancy data structure.
Suggestions:

1. Cartesian tree
2. Interval tree

Approach 1(offline)

1. There are n^2 possible intervals (not all of them are feasible).
2. If we can pre-process and store n^2 values and do give result in $\mathcal{O}(1)$, then it could be an interesting solution.
3. With n^2 preprocessing and $\mathcal{O}(nlog(n))$ binary search it is possible.

Approach 2(online)

1. Trying to solve the problem with a simple data structure. We can store the cities in a array.
2. We can make a tree above this array, for every pair of cities, make its parent one of the cities which has higher population. Similarly keep moving up the tree, by giving a value of the child which has higher population.

Build time : $\mathcal{O}(nlog(n))$
Space : $\mathcal{O}(n)$
Query time : $\mathcal{O}(log(n))$
Assumptions:

1. Assume that the number of cities are an integer power of 2, say 2^n. All the cities are the leaves of the tree.
2. The root of the tree is at height 0, and the leaves are at height n.
3. The nodes represent the population of the city, and are in the sorted order.

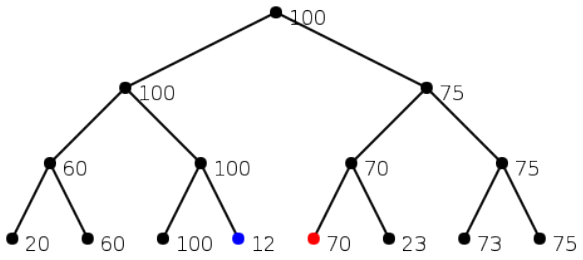

Figure 2:

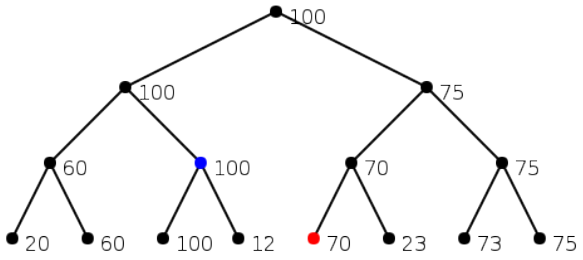

Figure 3:

4. The nodes are named as $a_{h,i}$, where h is the height and i is the position at that height from left. So the children of $a_{h,i}$ are $a_{h+1,2*i}$ and $a_{h+1,2*i+1}$.

5. Intermediate nodes (including the root), have the value which is maximum among their children.

Algorithm
Query: city with population p, at position between city i and $i+1$.

1. If $a_{n,i} > p$, then left bound is $a_{n,i}$.
   Else, keep moving to the parent, till you find a node whose value is $> p$, lets call it lh node.(lh = left higher)

2. If $a_{n,i}$ was not on the right sub-tree of the lh, move up the tree till you have $a_{n,i}$ on the right sub-tree. Rename the lh node to this node.

3. To find the leaf node which is the left bound, you keep moving down the left sub-tree of the lh node. At every level keep checking the values of the nodes, such that you remain on the sub-tree which has value closest to p, and greater than p.

Similar steps can be repeated to find the right bound city.
Figures 2 to 8, show steps to find the left and right bounds, for a particular example.

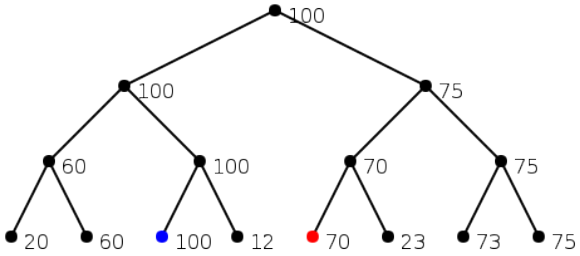

Figure 4:

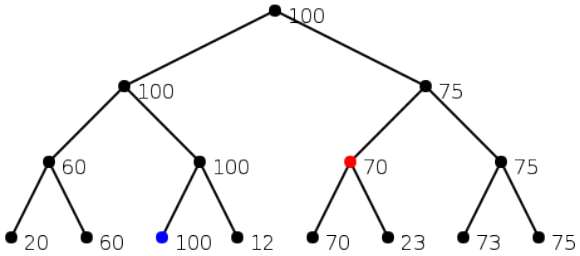

Figure 5:

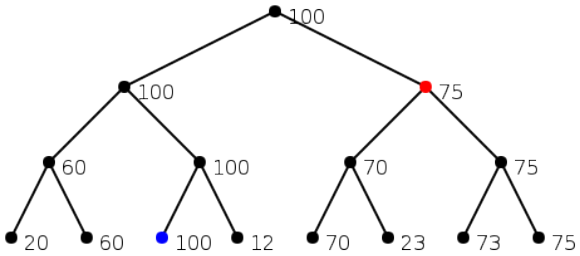

Figure 6:

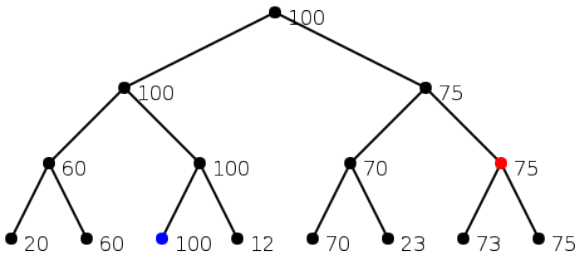

Figure 7:

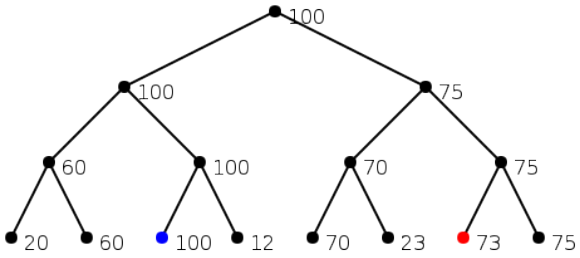

Figure 8: Left and right bounds for a city of size 72, in between position 3 and 4, i.e population 12 and 70.

Other interesting problems

1. For the 2D case, if the rectangles can shoot in different directions.
2. Instead of rectangular region, we can have circular region.
3. If instead of trying to find the optimum interval for a single city, if we try to find optimum position for several cities at once.