Lecture Notes: Range Counting

Yufei Tao
Department of Computer Science and Engineering
Chinese University of Hong Kong
taoyf@cse.cuhk.edu.hk

So far all the problems we have discussed are *reporting* problems, where we want to report the data elements qualifying a predicate. There is another class of problems that aim at outputting a constant amount of information to summarize a subset of elements. In this lecture, we will discuss such a problem called *range counting*. Let P be a set of N points in $\mathbb{R}^2$. Given an axis-parallel rectangle $q \in \mathbb{R}$, a *range count* query reports the number of points in P that are covered by q. The goal is to pre-process P into a data structure such that all queries can be answered efficiently.

Although we defined the problem by specifying q as a 4-sided rectangle, it is sufficient to focus on 2-sided q in the form of $(-\infty, x] \times (-\infty, y]$. A moment of thinking will reveal that the result of a 4-sided query can be obtained from four 2-sided queries.

We will assume that each (computer) word has at least $\log_2 N$ bits, which is a common assumption in the literature with good justifications. Most algorithms in computer science (including the ones to be presented below) implicitly assume that each real number or integer fits in one word. Notice that $\log_2 N$ bits are needed to represent N itself. So a CPU with word length less than this many bits cannot even represent the problem size (which is also the largest result size for a range query) in one word. Furthermore, the N input points can have different x-coordinates. Hence, the word length should be at least long enough so that we can represent N distinct x-coordinates, thus necessitating $\log_2 N$ bits.

We will introduce the *CRB-tree* [1] which uses $O(N/B)$ space, and answers a range count query in $O(\log_B N)$ I/Os. We will learn a technique that crams multiple integers into a single word—something that we have not attempted so far in this course. As usual, we will assume that P is in general position such that no two points in P share the same x- or y- coordinate.

1 Structure

The base tree of the CRB-tree is a B-tree $\mathcal{T}$ on the x-coordinates of the points in P. The leaf and branching parameters of $\mathcal{T}$ are both B. As before, given a node $u \in \mathcal{T}$, we define $\sigma(u)$ as the slab of u, and $sub(u)$ as its subtree. Also, let $P(u)$ be the set of points stored at the leaf nodes in $sub(u)$, and set $N(u) = |P(u)|$.

Consider u to be an internal node. We will associate u with a secondary structure $T(u)$ that consumes $O(\frac{N(u)}{B \log_B N})$ space, i.e., *sub-linear* to $N(u)$. As a result, the space of the secondary structures of all the nodes at each internal level of $\mathcal{T}$ adds up to $O(\frac{N}{B \log_B N})$. Since there are $O(\log_B N)$ internal level, the overall structure occupies $O(N/B)$ space.

Given two points p_1, p_2, we say that p_1 is *below* p_2 if the y-coordinate of the former is at most that of the latter. Suppose that u has f child nodes $u_1, ..., u_f$ (in ascending order of their slabs). Note that $P(u_1), ..., P(u_f)$ constitute a partition of $P(u)$. $T(u)$ will be used for the following purpose. Given an integer $r \leq N(u)$, we want to report, for *each* child node u_i ($1 \leq i \leq f$), how many points in $P(u_i)$ are below p_r, which is the r-th highest point of $P(u)$. In other words, f counts should be reported. Let us call this a *bundled probe*. We will see that $T(u)$ allows us to do so in $O(1)$ I/Os.

Let us break $P(u)$ into s subsets $P_u[1], ..., P_u[s]$, called *chunks*, along the y-dimension, i.e., points in $P_u[k]$ are all below those in $P_u[k']$ for any $k < k'$. Each chunk, except possibly the last one, has the same size $B \log_B N$. Hence, $s = O(\frac{N(u)}{B \log_B N})$. For each chunk $P_u[k]$, we keep a block of $f \leq B$ numbers $C_u[k][1], ..., C_u[k][f]$, where $C_u[k][i]$ equals the number of points from $P(u_i)$ in the preceding chunks, namely, $P_u[1], ..., P_u[k-1]$. Refer to $\{C_u[k][1], ..., C_u[k][f]\}$ as the *prefix count set* of $P_u[k]$. We store the prefix counts of all chunks in s consecutive blocks, so that the prefix counts of any $P_u[k]$ can be fetched in one I/O.

In a bundled probe with input r, the chunk that contains p_r is $P_u[k^\star]$ where $k^\star = \lceil r/(B \log_B N) \rceil$. The prefix count set of $P_u[k^\star]$ tells us, for each child node u_i, how many points from $P(u_i)$ are in the first $k^\star - 1$ chunks. Hence, we still have to know exactly how many of the lowest $r - k^\star B \log_B N$ points in $P_u[k^\star]$ are from $P(u_i)$.

For this purpose, we pre-compute some extra information about $P_u[k^\star]$. For each point in chunk $P_u[k^\star]$, we use $\log_2 B$ bits to record its *branch index*, i.e., which of the $f \leq B$ child nodes has p in the subtree. As $P_u[k^\star]$ has $B \log_B N$ points, their branch indexes require $B \cdot \log_B N \cdot \log_2 B = B \log_2 N$ bits in total, which is exactly how many bits a block has. Hence, we can keep all of them in one block. Care should be exercised to put the branch index of the lowest point of $P_u[k^\star]$ as the first $\log_2 B$ bits in this block, that of the second lowest point as the second $\log_2 B$ bits, and so on. In this way, a bundled probe with input r can know which bits belong to the lowest $r - k^\star B \log_B N$ points. Note that after reading the branch index block, bit inspection is done in memory, and thus incurs no cost. We put the branch index blocks of all chunks consecutively, so that the block of any chunk can be obtained in one I/O by chunk id.

Overall, $T(u)$ contains two blocks for each chunk: one for its prefix counter set, and the other as its branch index block. It therefore uses $s = O(\frac{N(u)}{B \log_B N})$ space.

2 Query

We proceed to explain how to process a range count query with search region $q = (-\infty, x] \times (-\infty, y]$. First, we obtain the number r_y of points in P whose y-coordinates are at most y. This can be done in $O(\log_B N)$ I/Os using a slightly-augmented B-tree (think: how?). Then, initialize the query result $res = 0$, and invoke the following $probe(u, r)$ function, by setting u to the root of $\mathcal{T}$ and $r = r_y$.

In general, we keep to the invariant that, every time $probe(u, r)$ is invoked, r equals the number of points in $P(u)$ whose y-coordinates are at most y. The function returns the number of points in $P(u) \cap q$. If u is a leaf node, we accomplish this by simply checking every point in u. Otherwise, assume that u has child nodes $u_1, ..., u_f$. We first obtain the α such that $x \in \sigma(u_\alpha)$. Then, we perform on $T(u)$ a bundled probe with input r. Recall that the bundled probe returns f values $\beta_1, ..., \beta_f$, where β_i equals the number of points in $P(u_i)$ that are under the r-th highest point in $P(u)$, or equivalently (by the definition of r), that have y-coordinates at most y. We increase res by $\sum_{i=1}^{\alpha-1} \beta_i$ and then by the value returned from $probe(u_\alpha, \beta_\alpha)$.

Overall, the query algorithm performs constant I/Os at each node along a root-to-leaf path of $\mathcal{T}$. The query cost is therefore $O(\log_B N)$.

References

[1] S. Govindarajan, P. K. Agarwal, and L. Arge. CRB-tree: An efficient indexing scheme for range-aggregate queries. In *ICDT*, pages 143–157, 2003.