Lecture Notes: External Range Tree

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

This lecture will discuss the *range searching* problem. Let P be a set of N points in $\mathbb{R}^2$. Given an axis-parallel rectangle q, a *range query* reports all the points of $P \cap q$. We want to maintain a dynamic structure on P to answer range queries efficiently.

We will introduce the *external range tree* [1] which solves a query in $O(\log_B N + K/B)$ I/Os, where K is the number of points reported. It uses $O(\frac{N}{B}\frac{\log N}{\log\log_B N})$ space and can be updated in $O(\log_B N\frac{\log N}{\log\log_B N})$ I/Os amortized. One may wonder whether the space can be improved (say to linearity) while still achieving the query cost $O(\log_B N + K/B)$ (which, by the way, can be shown to be optimal using advanced techniques beyond this course). Unfortunately, this has been proved to be impossible. The space consumption of the external range tree is already asymptotically the lowest to achieve this query time [2].

1 Structure

The external range tree combines almost all the structures we have discussed previously: the weight-balanced B-tree, the external interval tree (and hence, also the persistent B-tree), and the external priority search tree. The base structure is a weight-balanced B-tree $\mathcal{T}$ on the x-coordinates of P. The leaf parameter of $\mathcal{T}$ is B, and its branching parameter is set to $\Theta(\log_B N)$. Note that this is the first time we see a weight-balanced B-tree whose branching parameter depends on N. The height of the tree is $O(\log_{\log_B N}(N/B)) = O(\frac{\log N}{\log\log_B N})$.

As before, each node u of $\mathcal{T}$ naturally corresponds to a vertical slab $\sigma(u)$ in $\mathbb{R}^2$. We denote by $P(u)$ the set of points whose x-coordinates are stored in the subtree of u. In other words, $P(u) = \sigma(u) \cap P$. If u is a leaf node, $P(u)$ is associated with u and stored in $O(1)$ blocks.

Each internal u is associated with several secondary structures. Let $u_1, ..., u_f$ be the child nodes of u. For each child node u_i, associate with u two external priority search trees $\Pi_{\sqsupset}(u, u_i)$ and $\Pi_{\sqsubset}(u, u_i)$ on $P(u_i)$, which support 3-sided range queries whose search regions have the shapes $\sqsupset$ and $\sqsubset$, respectively. Furthermore, for each $i \in [1, f]$, create a B-tree $\Xi(u, u_i)$ on the points of $P(u_i)$ indexing their y-coordinates. Finally, u is also associated with an external interval tree $\Gamma(u)$ built on a set $S(u)$ of 1d intervals obtained as follows. For each $i \in [1, f]$, let $Y_1, ..., Y_n$ be the y-coordinates of the points of $P(u_i)$ in ascending order, where $n = P(u_i)$. Generate a set $S_i(u)$ of n 1d intervals where the j-th ($1 \leq j \leq n$) equals $(Y_{j-1}, Y_j]$ (defining a dummy $Y_0 = -\infty$). The interval carries a pointer to the leaf node of $\Xi(u, u_i)$ where the point corresponding to Y_j is stored, so that once $(Y_{j-1}, Y_j]$ is retrieved, we can jump to that leaf node in 1 I/O. $S(u)$ is the union of $S_1(u), ..., S_f(u)$.

Clearly, the space of all the secondary structures of an internal node u is $O(|P(u)|/B)$. Hence, the secondary structures of all the nodes at an internal level of $\mathcal{T}$ occupy $O(N/B)$ space in total. The overall space consumption of the external range tree is therefore $O(\frac{N}{B}\frac{\log N}{\log\log_B N})$.

Supporting an insertion in $O(\log_B N\frac{\log N}{\log\log_B N})$ amortized I/Os should be quite straightforward

at this stage of the course, by observing that all secondary structures are easy to update and utilizing the properties of the weight-balanced B-tree in a standard way. Resorting to global rebuilding, we can handle a deletion easily in the same time bound.

2 Query

Given a query with search region $q = [x_1, x_2] \times [y_1, y_2]$, we first find the lowest node u whose slab contains q. Then, we will find all the K result points using only the secondary structures of u in $O(\log_B N + K/B)$ I/Os.

We will discuss only the case where u is an internal node. Let $u_1, ..., u_f$ be the child nodes of u. Suppose that x_1 (x_2) falls in some $\sigma(\alpha)$ ($\sigma(\beta)$), where $1 \leq \alpha < \beta \leq f$. Clearly, only the points in $P(u_\alpha)$, $P(u_{\alpha+1})$, ..., $P(u_\beta)$ can possibly fall in q. We find $P(u_\alpha) \cap q$ by performing a 3-sided range query on $\Pi_{\sqsubset}(u, u_\alpha)$ with the search region $q \cap \sigma(u_\alpha)$, and similarly, find $P(u_\beta) \cap q$ by performing a 3-sided range query on $\Pi_{\sqsupset}(u, u_\beta)$. The cost of the two queries is $O(\log_B N + K_1/B)$ where K_1 is the number of points retrieved by them.

It remains to report the qualifying points in $P(u_{\alpha+1})$, $P(u_{\alpha+2})$, ..., $P(u_{\beta-1})$. For this purpose, we search $\Gamma(u)$ with the value y_1, which reports at most one interval from each $S_i(u)$ where $1 \leq i \leq f$. The cost is $O(\log_B N + f/B) = O(\log_B N)$ I/Os. Now consider each $i \in [\alpha + 1, \beta - 1]$, and let $(Y_{j-1}, Y_j]$ be the interval fetched from $S_i(u)$. Then, the point p that corresponds to Y_j is the lowest point in $P(u_i)$ on or above the horizontal line $y = y_1$. We jump to the leaf node of $\Xi(u, u_i)$ storing p in $O(1)$ I/O, and then scan in $\Xi(u, u_i)$ the points of $P(u_i)$ in ascending order of their y-coordinates starting from p, until seeing the first point with a coordinate greater than y_2. All the other points scanned are reported. In total, we spend $O(\log_B N + f + K_2/B) = O(\log_B N + K_2/B)$ I/Os reporting the points in $P(u_{\alpha+1})$, $P(u_{\alpha+2})$, ..., $P(u_{\beta-1})$, where K_2 is the number of these points.

So far the total cost is $O(\log_B N + K_1/B + K_2/B) = O(\log_B N + K/B)$. Note that we have left out an important detail – how to find u in the first place? Naively, this can be done by traversing at most a single root-to-leaf path in $O(\frac{\log N}{\log \log_B N})$ I/Os, which, however, is not necessarily $O(\log_B N)$. We will fix this later using a *path packing* method.

3 Path Packing

As mentioned earlier, the height $O(\frac{\log N}{\log \log_B N})$ of $\mathcal{T}$ may be $\omega(\log_B N)$, which makes it too expensive for us to access a complete root-to-leaf path of $\mathcal{T}$. Next, we describe a trick to remedy this issue. We will assume $\log_B N \leq B^{1/3}$; otherwise, $O(\frac{\log N}{\log \log_B N}) = O(\log_B N)$.

Since our goal is to identify efficiently the lowest node whose slab contains a query rectangle, we can focus on just the base tree $\mathcal{T}$ and ignore all the secondary structures. Let H be the height of $\mathcal{T}$ and h the largest integer such that $\sum_{i=0}^{h} (\log_B N)^i \leq B$. Notice that $h \geq 2$ and $h = \Omega(\frac{1}{\log_B \log_B N})$. The observation behind path packing is that using a full block to store an internal node u is wasteful, because u has at most $\log_B N \ll B$ entries. In fact, we can pack as many as $\sum_{i=0}^{h} (\log_B N)^i$ nodes in $O(1)$ blocks. A careful look at $\sum_{i=0}^{h} (\log_B N)^i$ reveals that this is essentially the number of nodes in a perfect $(\log_B N)$-ary tree with $h + 1$ levels. The observation motivates us to pack multiple nodes of $\mathcal{T}$ into $O(1)$ blocks.

To keep everything conceptually clean, we will leave $\mathcal{T}$ untouched but duplicate it into $\mathcal{T}'$, except that the nodes of $\mathcal{T}'$ are stored in a compact manner as follows. Given a node u of $\mathcal{T}$, let $sub(u, h)$ be the part of $\mathcal{T}$ that includes u and all its descendants up to h levels below (i.e., if u is

at level x, then $sub(u,h)$ includes its descendants at level $x-1, ..., x-h$). Call $sub(u,h)$ the *h-level subtree* of u.

We pack $\mathcal{T}$ into $\mathcal{T}'$ as follows. First, let u be the root of $\mathcal{T}$. In $\mathcal{T}'$, $sub(u,h)$ is stored together in constant blocks. Recursively, for each leaf node v of $sub(u,h)$, we store $sub(v,h)$ in $\mathcal{T}'$ also using $O(1)$ blocks, and repeat this until the leaf level of $\mathcal{T}$ has been reached. In this way, any root-to-leaf path of $\mathcal{T}$ is stored in $O(H/h) = O(\frac{\log N}{\log\log_B N} \cdot \log_B \log_B N) = O(\log_B N)$ blocks in $\mathcal{T}'$. In other words, a root-to-leaf path can now be traversed in logarithmic I/Os.

Is the maintenance of $\mathcal{T}'$ more expensive than that of $\mathcal{T}$? Fortunately, no. Clearly, $\mathcal{T}'$ does not need to be modified unless a node u in $\mathcal{T}$ is split. In that case, we can simply rebuild the entire subtree of u (i.e., the h-level subtree $sub(u,h)$ of u, those of the leaf nodes in $sub(u,h)$, and recursively on) in $O(w(u)/B))$ I/Os. In other words, each insertion accounts for only $O(H) = O(\frac{\log N}{\log\log_B N})$ I/Os of the maintenance of $\mathcal{T}'$.

We conclude:

Theorem 1. *There is a structure on a set of N points in $\mathbb{R}^2$ that occupies $O(\frac{N}{B}\frac{\log N}{\log\log_B N})$ space, answers a range query in $O(\log_B N + K/B)$ I/Os, and supports an update in $O(\log_B N \frac{\log N}{\log\log_B N})$ amortized I/Os.*

References

[1] L. Arge, V. Samoladas, and J. S. Vitter. On two-dimensional indexability and optimal range search indexing. In *PODS*, pages 346–357, 1999.

[2] J. M. Hellerstein, E. Koutsoupias, D. P. Miranker, C. H. Papadimitriou, and V. Samoladas. On a model of indexability and its bounds for range queries. *JACM*, 49(1):35–55, 2002.