# The van Emde Boas Structure

[Notes for ESTR2102]

Yufei Tao

CSE Dept
CUHK

We have already learned that a predecessor can be found in $O(\log n)$ time after suitable preprocessing. Today, we will derive another bound when the underlying set consists of only integers in the domain $[1, U]$. Our new structure—called the van Emde Boas (vEB) structure—achieves the query time of $O(\log \log U)$.

## Predecessor Search

Let $S$ be a set of $n$ integers, each of which comes from the domain $[1, U]$. We want to store $S$ in a data structure to support:

- A predecessor query: give an integer $q$, find its predecessor in $S$, which is the largest integer in $S$ that does not exceed $q$.

We will assume that $U = 2^{2^\alpha}$ for some integer $\alpha \geq 0$. This assumption is made without loss of generality (this will be obvious, and will be left to you).

## vEB-Structure

We will describe the vEB-structure in a recursive manner.

**Base Case:** If $U = 2$, we simply store $S$ in a linked list.

## vEB-Structure

**General Case:** Now consider that $U > 2$.

We divide the universe $[1, U]$ into disjoint segments, each of which has length $\sqrt{U}$. Note that by our assumption that $U = 2^{2^\alpha}$, $\sqrt{U}$ is always an integer.

We can therefore label the segments from left to right with ids 1, 2, ..., $\sqrt{U}$. If a segment contains at least one integer of $S$, we say that the segment is non-empty; otherwise, it is empty.

For every non-empty segment $\sigma$, denote by $S(\sigma)$ the set of integers of $S$ covered by $\sigma$.

vEB-Structure

**General Case (cont.):**

**Structure 1:** Let $B$ be the set of non-empty segments' ids. Build a hash table $H$ to answer the following query: given an integer $i \in [1, \sqrt{U}]$, is $i \in B$?

**Structure 2:** Store with each non-empty segment $\sigma$ the largest integer in $S(\sigma)$, which is denoted as $max(s)$. Store also the largest integer in the non-empty segment immediately preceding $\sigma$, which is denoted as $leftmax(s)$.

## vEB-Structure

**General Case (cont.):** Now here comes the recursive part.

**Structure 3:** Build a vEB-structure to answer predecessor queries on $B$ in the universe $[1, \sqrt{U}]$.

**Structure 4:** Each non-empty segment $\sigma$ defines a universe of its own with length $\sqrt{U}$. Build a vEB-structure to answer predecessor queries on $S(\sigma)$ in that universe.

> Note that the recursion eventually ends because the universe keeps shrinking.

## Query

Let us now discuss how to answer a query with search value $q$.

First, obtain the id $x$ of the segment containing $q$: $x = \lceil q/\sqrt{U} \rceil$. Then, do dictionary search on $H$ to find out whether $x \in B$.

- If no: it means that segment $x$ is empty. We know that the predecessor of $q$ equals $max(\sigma)$, where $\sigma$ is the non-empty segment whose id is the predecessor of $x$ in $B$. Hence, solve the query by performing predecessor search on Structure 3.
- If yes: then segment $x$ is non-empty—denote it by $\sigma$. Obtain $leftmax(\sigma)$. Find the predecessor $y$ of $q$ on $S(\sigma)$ (recursively using Structure 4). If $y$ exists, it is the final answer; otherwise, the final answer is $leftmax(\sigma)$.

## Query Time Analysis

Let $f(U)$ be the time of a query when the universe has length $U$.

Searching $H$ takes $O(1)$ time (use perfect hashing to achieve worst case). In either the yes or the no case, we do one query in a smaller universe of length $\sqrt{U}$. Hence:

$$f(U) \leq O(1) + f(\sqrt{U})$$

with the terminating condition that $f(2) = O(1)$.

Solving the recurrence gives $f(U) = O(\log \log U)$ (worst case).

## Space Analysis

Let $g(n, U)$ be the space of a van Emde Boas structure of $n$ elements in a universe of length $U$.

Structures 1 and 2 obviously occupy only $O(n)$ space. Structure 3 takes $g(n, \sqrt{U})$ space. Regarding Structure 4, suppose that we have $t$ non-empty segments, covering $n_1, n_2, ..., n_t$ integers of $S$, respectively ($\sum_{i=1}^{t} n_i = n$). We know that the vEB-structure on the $i$-th ($1 \leq i \leq t$) segment requiress $g(n_i, \sqrt{U})$ space. Hence:

$$g(n, U) \quad \leq \quad O(n) + g(n, \sqrt{U}) + \sum_{i=1}^{t} g(n_i, \sqrt{U})$$

with the terminating condition that $g(n, U) = O(1)$ when either $n$ or $U$ is at most a constant.

Solving the recurrence gives $g(n, U) = O(n \log U)$.

Next, we will reduce the space to $O(n)$, without affecting the query time, using a technique called bootstrapping.

## Bootstrapping

Sort all the integers of $S$. Divide $S$ into disjoint intervals, each of which covers $\log_2 U$ integers of $S$, except possibly the last one. There are $O(n/\log U)$ intervals.

Create a set $S'$ by taking the smallest integer of $S$ in each interval.

For each interval, create a binary search tree (BST) on the at most $\log_2 U$ integers therein.

Create a vEB-structure on $S'$.

> Overall space is now $O(n)$! Note that the vEB-structure on $S'$ takes $O(\frac{n}{\log U} \log U) = O(n)$ space.

## Bootstrapping

Now let us see how to answer a query with search value $q$.

First, find the predecessor $x$ of $q$ in $S'$. This takes $O(\log \log U)$ time using the vEB-structure.

Then, go to the interval containing $x$, and find the predecessor of $q$ within that interval. This takes $O(\log \log U)$ time using the BST of that interval—recall that the BST stores only $O(\log U)$ elements.

The overall query time is therefore $O(\log \log U)$.