Contents

PART I

DataStructures

CHAPTER 1

Lecture I

1.1 The Static Predecessor problem

Data structure is represented by *Set* S of items $\{x_1, .., x_n\}$
Query : $\text{pred}(z) = \max\{\, x \in S : x \leftarrow z \,\}$
Example Solution : $\mathcal{O}(n \log n)$ using a balanced BST

Work RAM Model

- Item are integers $\{0, 1, \ldots, 2^w - 1\}$
- "w" word size is$u = 2^w - 1$
- Pointers fit in a word
- Space $\geq n$ and $w \geq \log(\text{Space}) \geq \log(n)$

Two Data Structures

- Van Emde Boas Tree FOCS '75
  - Update/Query $\mathcal{O}(\log w)$ time
  - Update/Query $\mathcal{O}(\log u)$ space
- y Fast Tries Willard '83

Fusion Trees Fredman,Willard JCCS '93

- Query is $\mathcal{O}(\log w^n)$ time
- Can achieve $\min(\lg w, \log w^n) \leq$

- Dynamic Fusion Trees $\mathcal{O}(n \lg n)$ sorting
- Dynamic Fusion Trees can also achieve $\mathcal{O}(n \lg \lg n)$ sorting (Han STOC '02) and $\mathcal{O}(n\sqrt{\lg \lg n})$ FOCS '02

Van Emde Boas

- Fields of $VEB_u(V)$
  - $\sqrt{u}$ size array V.cluster[0] ... V.cluster$\sqrt{u}$
  - V.summary is a VEB $\sqrt{u}$
  - V.min and V.max are integers in {0, ..., u-1}
- Represention of values
  - X $\in$ {0,1, ..., u-1}
  -

$$x = \underbrace{1011}_{\mathbf{c}}\overbrace{0011}^{\mathbf{i}}$$

Predecessor Algorithm $(V, x = \langle c, i \rangle)$

$$\begin{aligned}
&\text{if } x > V.\max: \\
&\quad \text{return } V.\max \\
&\text{else if } V.\text{cluster}[c].\min < x: \\
&\quad \text{return } \text{pred}(V.\text{cluster}[c], i) \\
&\text{else } c' = \text{pred}(V.\text{summary}, c): \\
&\quad \text{return } V.\text{cluster}[c'].\max
\end{aligned}$$

Insertion Algorithm $(V, x = \langle c, i \rangle)$

$$
\begin{aligned}
&\text{if } V = \varnothing : \\
&\quad V.\min \leftarrow x^{prime}; \\
&\text{if } x < V.\min : \\
&\quad swap(x, V.\min); \\
&\text{else if } V.\text{cluster}[c].\min = \varnothing : \\
&\quad insert(V.summary, c); \\
&\text{insert}(V.\text{cluster}[c], i)
\end{aligned}
$$

Algorithm analysis $(V, x = \langle c, i \rangle)$

- Prediction time
  - T (u) = T ($\sqrt{u}$) + $\mathcal{O}(1)$
  - T (u) = $\mathcal{O}(\lg \lg u) = \mathcal{O}(\lg w)$
- Insertion time
  - $T(u) \leq 2T(\sqrt{u}) + \mathcal{O}(1)$
  - $T(w) \leq 2T\mathcal{O}\left(\frac{w}{2}\right) + \mathcal{O}(1)$

Space of VEB

$S(u) = (\sqrt{u} + 1)S(\sqrt{u}) + \mathcal{O}(1) \rightarrow S(u) = \theta u$

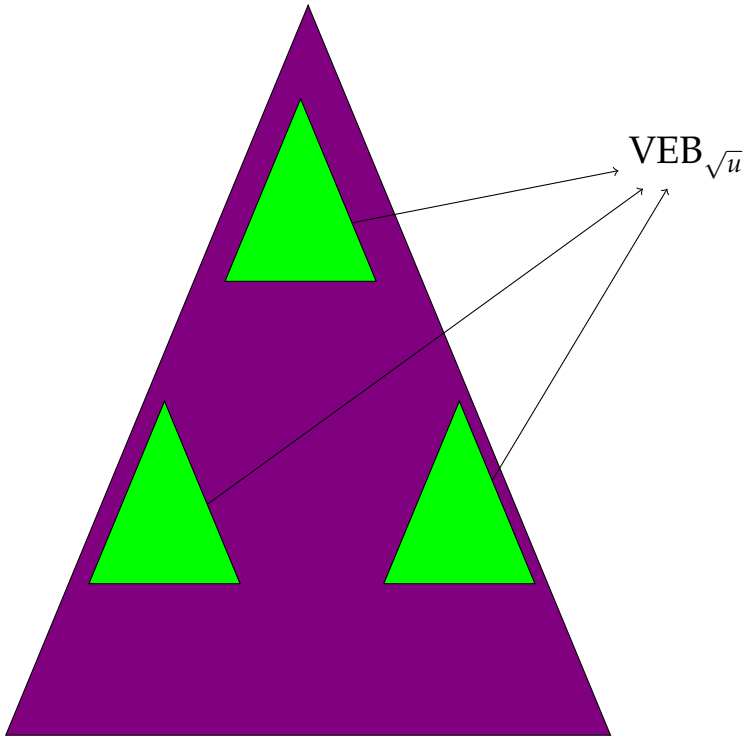

Figure 1.1: Van Ende Boas

Binary trie

Search time is $\mathcal{O}(\log n)$

- Red dashed lines are jump Pointers
- Jump pointers are used when children are missing to point to the successor

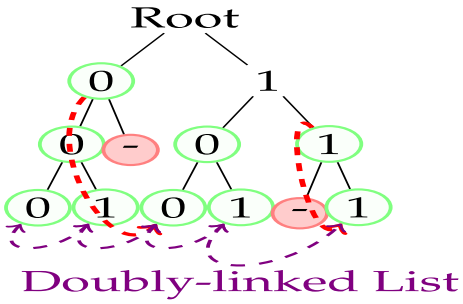

Figure 1.2: Binary Trie

Add node to Binary trie

Insertion time is $\mathcal{O}(\log n)$

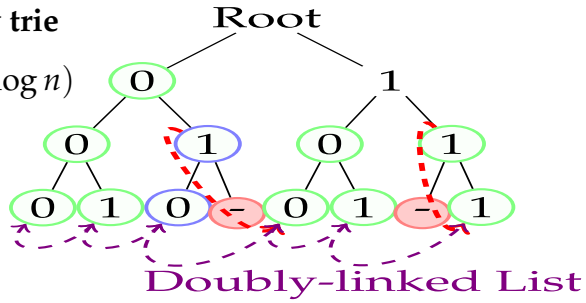

Figure 1.3: Binary Trie; Add Node

This shows an example of addition of a new node. The New node and its left child are in blue. The jump pointers are updated. And new node is added to the doubly-linked list at the appropriate position

The integer inserted above is

$$0\,1\,0$$

Remove node from Binary trie

Removal time is $\mathcal{O}(\log n)$

- Remove node (It will be a leaf)
- Walk back up and remove any node without a child and The jump pointers are updated. The doubly-linked list is updated.

x-fast trie

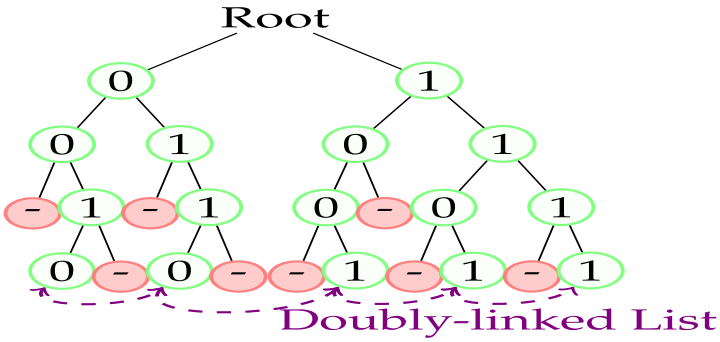

Figure 1.4: xfasttrie

- An x-Fast Trie is a threaded binary trie where leaves are stored in a doubly-linked list and where all nodes in each level are stored in a hash table.
- Can do lookups in the HashTable in time $\mathcal{O}(1)$.
- Claim: Can determine successor(x) in time $\mathcal{O}(\log \log u)$.
- Start by binary searching for the longest prefix of x.
- If at a leaf node, follow the forward pointer to the successor.
- If at an internal node with a missing 1, follow the 1 thread.
- If at an internal node with a missing 0, follow the 0 thread and follow the forward pointer.

Fusion tries - Fredman, Willard JCSS '93

- Static Predecessor
- Made dynamic by Andersson, Thorup JACM $(\log_w n \log \log n)$ deterministic updates

- Made dynamic by Raman ESA JACM '96 $\mathcal{O}(\log_w n)$ updates (Expected time since it is a randomized algorithm)
- Query for all $\mathcal{O}(\log_w n)$

Idea : B-Tree with branching factor $w^1/5$

- Fusion tree node stores k = $\mathcal{O}(w^{1/5})$ keys
- Item are integers $\{x_0, \ldots, x_{k-1}\}$
- $\mathcal{O}(1)$ Predecessor/Successor
- Polynomial time $k^{0(1)}$ Preprocessing

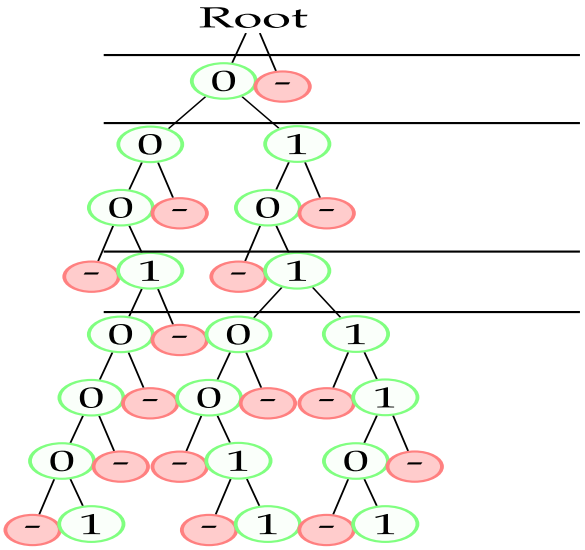

Figure 1.5: Fusion Trees Distinguishing Keys

The tree shown has

k-1

branching nodes and the height of the tree is

w

$\rightarrow \leq$ k-1 levels containing a branching node. The levels starting from the top of the tree correspond to important bits the indices of which are $b_0, \ldots, b_r$ where $r < k = \mathcal{O}(w^{1/5})$ The leaves have the least significant bits.

The parallel lines show that we can check some branching nodes to find out if the lower bits can exist or not instead of traversing upto the leaf. Refer the section on Sketching.

What is sketching in this algorithm?

sketch(x) = Extract bits $\{b_0, \ldots, b_{r-1}\}$

$$1\underbrace{1}_{b_3}1\underbrace{0}_{b_2}\underbrace{0}_{b_1}01\underbrace{1}_{b_0}$$

Sketching Example

- The important bits (violet circles) are used for sketching
- The labels of the leaf nodes also show these (violet) bits

The query is shown by the blue path.

$$\underbrace{0101}_{\text{Query}}$$

The sketch is

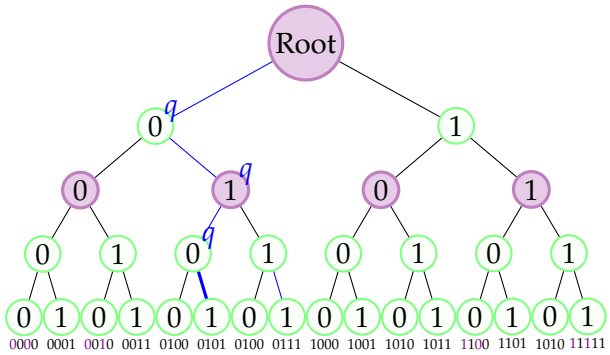

Figure 1.6: Example of Sketching

$$\underbrace{0}_{}\,0\,\underbrace{0}_{}\,0$$
$$\underbrace{0}_{}\,0\,\underbrace{1}_{}\,0$$
$$\underbrace{1}_{}\,1\,\underbrace{0}_{}\,0$$
$$\underbrace{1}_{}\,1\,\underbrace{1}_{}\,1$$

Since the sketch of the query 0101 matches the sketch 00 the answer will be the left-most node in the tree. This is wrong. It is not the predecessor 0010 or the successor 1100

We need to find a solution for this.

Let us assume sketch(x) $\leq$ sketch(g) $\leq$ sketch(x_{k+1}). We will look for the longest common prefix/ancestor of the query

q

and either x_i or x_{i+1}.

Now we can identify the wrong path we took. The correct path from the LCA(Least common Ancestor) would have lead to the left, to the tree whose nodes are shown in grey. Identify the maximum in that tree. This is e. And find sketch(e) among sketch(x_i). Now we can find the predecessor sketch(e) among sketch(x_i). Similarly we find the successor using the same algorithm.

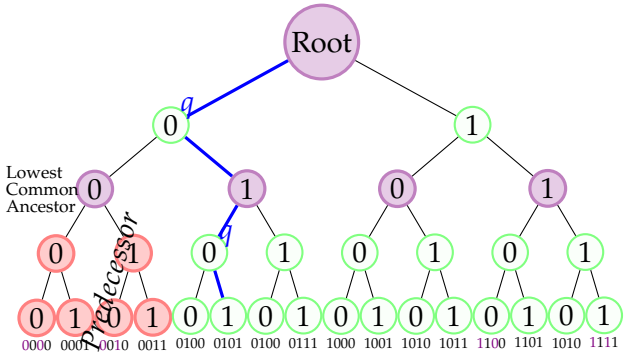

Figure 1.7: Example of Sketching

Approximate sketch(x)

We can spread the important bits in predictable pattern of len($\mathcal{O}(w^{1/5})$)
Idea : Mask important bits

x′ = x and $(\sum_i^{r-1} 2^{b_i})$ and multiple x′.m = $(\sum_i^{r-1} x_{b_i} 2^{b_i})((\sum_j^{r-1} 2^{m_j}))$ = $(\sum_i^{r-1} \sum_j^{r-1} 2^{b_i+m_j}$

Claim : for any $b_0 \dots b_{r-1}$ we can choose for any $m_0 \dots m_{r-1}$ such that

1. $b_i + m_j$ are distinct $foralli, j$
2. $b_0 + m_0 < \cdots < b_{r-1} + m_{r-1}$
3. $(b_{r-1} + m_{r-1}) - (b_0 + m_0) = \mathcal{O}(r^4) = \mathcal{O}(w^{4/5})$

Proof

Choose $m'_0, \dots, m'_{r-1} < r^3$ (Start with Stronger assumption) such that $b_i + m'_j <$ mod r^3

Property 1:

Pick $m'_0, \dots, m'_{t-1}$ by induction. m'_t must avoid $m'_i + b_j + b_k$ mod r^3, $\forall$ $\underbrace{i}_{\text{t}}, \underbrace{j}_{\text{r}}, \underbrace{k}_{\text{r}}$

$\leftarrow$ $tr^2 < r^3$

Property 2:

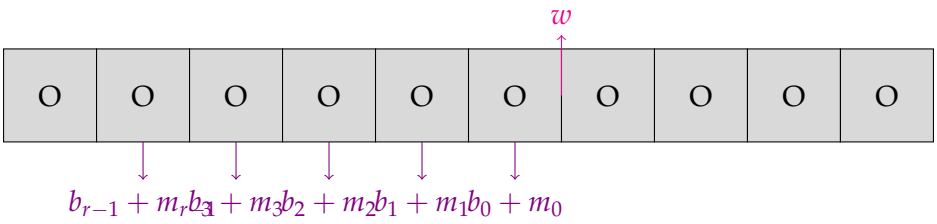

Figure 1.8: Description of Proof

Pick $m_i = m'_i + (w - b_i + i_{r^3})$ rounded down to r^3
We also need to mask with $(\sum_i^{r-1} 2^{b_i+m_i}$ and shift by $(b_0 + m_0$

CHAPTER 2

Lecture II

2.1 Order Statistics problem

Given k elements in an array, find kth smallest element (element of k rank)

Sorting will take $\mathcal{O}(n \log n)$ and we want to better than that, ideally better than linear time.

We know that that median is $\left\lceil \frac{k+1}{2} \right\rceil$ or $\left\lfloor \frac{k+1}{2} \right\rfloor$

It is tricky to find the median in linear time. In addition we also find kth smallest element.

That is the focus of this lecture.

Randomized Divide and Conquer

Rand-select $(A - p - q - \underbrace{\qquad i \qquad}_{i\text{th smallestelement in } A}$ to

$$\begin{aligned}
&\text{if } p = q \text{ then return A[p] :}\\
&r \leftarrow \text{Rand-partition } A - p - q\\
&k \leftarrow \text{r - p - 1}\\
&\text{else if } V.\text{cluster}[c].\text{min} < x :\\
&\quad \text{return pred}(V.\text{cluster}[c], i)\\
&\text{else } c' = \text{pred}(V.\text{summary}, c) :\\
&\quad \text{return } V.\text{cluster}[c'], \text{max}
\end{aligned}$$

Rand-partition A - p - q means that we pick a value in the range p-q and use it partition the array, so that all elements less than the *random* elements are the left and the greater elements are on the right

CHAPTER 3

Lecture I/II/III

3.1 Basic Definitions

- state s_t, is the state of the world at time t
- action a_t, the decision taken at time t
- trajectory τ, sequence of states/obserations and actions D $:= (s_1, a_1, \ldots, s_T)$
- reward function $r(s, a)$
- Policy $\pi(a, s)$ or $\pi(a, o)$

$$\min_{\theta} \mathbb{E}_{\tau \sim \mathrm{P}\cdot(\text{ø})} \left[\sum_{t}^{T} r(s_t, a_t) \right]$$

3.2 Imitation Learning

What is the goal of imitation learning ?

- Data: Given trajectories collected by an expert "demonstrations" D $:= (s_1, a_1, \ldots, s_T)$
- Sampled from some unknown policy π_{expert}
- Goal: Learn a policyπ_θ that performs at the level of π_{expert} by mimicking

Imitation learning -Version 0

Deterministic policy

- Data: Given trajectories collected by an expert "demonstrations"D $:= (s_1, a_1, \ldots, s_T)$

- Supervised regression to the expert's policy $\min_{\theta} \frac{1}{\|\mathrm{D}\|} \sum_{(s,a)\in \mathrm{D}} \|a - \hat{a}\|_2^2$
- Deploy learned policy π_{theta}

Evaluation of the RL objective

$$\theta^* = \text{argmax}_\theta \underbrace{\mathbb{E}_{\tau\sim \mathrm{P}(\text{ø})} \left[\sum_t^T r(s_t, a_t) \right]}_{J(\theta)}$$

$$\nabla_\theta J(\theta) = \nabla_\theta \int P_\theta(\tau) r(\tau) d\tau = \int \nabla_\theta P_\theta(\tau) r(\tau) d\tau$$

Another paragraph with title

This gradient can be simplified using this trick.

$= P_\theta(\tau) \nabla_\theta \log P_\theta(\tau)$

$= P_\theta(\tau) \frac{\nabla_\theta P_\theta(\tau)}{P_\theta(\tau)}$

We apply the chain rule after evaluating $\log P_\theta(\tau)$ which is $\frac{1}{P_\theta(\tau)}$