

# Contents

<b>I</b>	<b>DataStructures</b>	<b>2</b>
<b>1</b>	<b>Lecture I</b>	<b>3</b>
1.1	The Static Predecessor problem . . . . .	3
<b>2</b>	<b>Lecture II</b>	<b>14</b>
2.1	Order Statistics problem . . . . .	14
<b>3</b>	<b>Lecture I/II/III</b>	<b>16</b>
3.1	Basic Definitions . . . . .	16
3.2	Imitation Learning . . . . .	16

**PART I**

# *DataStructures*

## CHAPTER 1

# *Lecture I*

### 1.1 The Static Predecessor problem

Data structure is represented by *Set*  $S$  of items  $\{x_1, \dots, x_n\}$

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  $\{0, 1, \dots, 2^w - 1\}$
- “w” word size is  $u = 2^w - 1$
- Pointers fit in a word
- $\text{Space} \geq n$  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  $V.\text{cluster}[0] \dots V.\text{cluster}_{\sqrt{u}}$
  - $V.\text{summary}$  is a  $VEB_{\sqrt{u}}$
  - $V.\text{min}$  and  $V.\text{max}$  are integers in  $\{0, \dots, u-1\}$
- Representation of values
  - $X \in \{0, 1, \dots, u-1\}$
  -

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

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

```

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

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

```

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

```

**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**

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

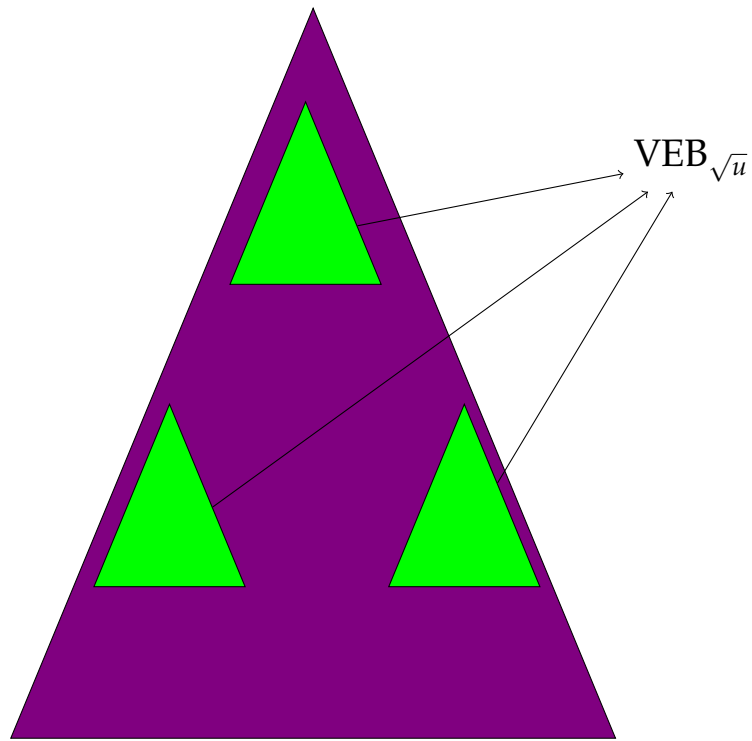


Figure 1.1: Van Ende Boas

## 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

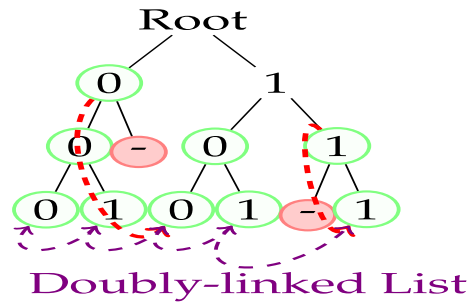


Figure 1.2: Binary Trie

### Add node to Binary trie

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

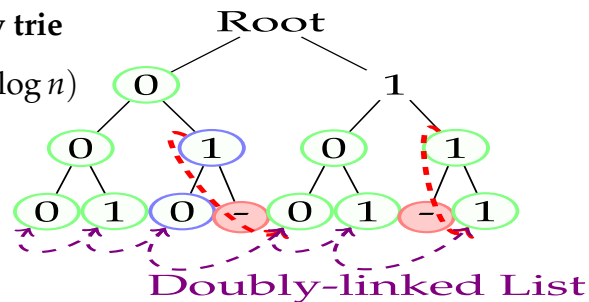


Figure 1.3: Binary Trie; Add Node

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

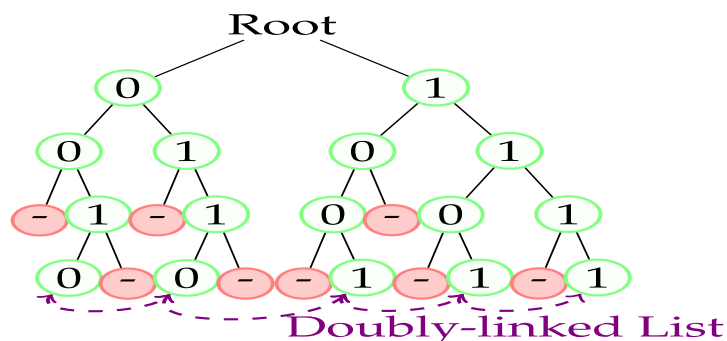


Figure 1.4: xfasttrie

- 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  $\text{successor}(x)$  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
- Items are integers  $\{x_0, \dots, x_{k-1}\}$
- $\mathcal{O}(1)$  Predecessor/Successor
- Polynomial time  $k^{0(1)}$  Preprocessing

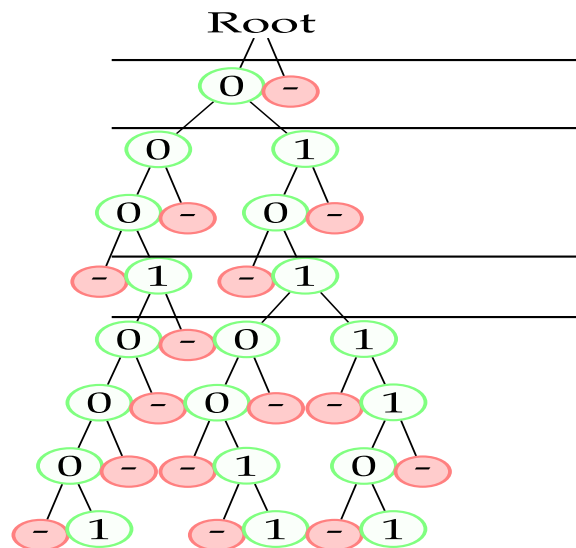


Figure 1.5: Fusion Trees Distinguishing Keys

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  $b_0, \dots, b_r$  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  $\{b_0, \dots, b_{r-1}\}$

$$\begin{array}{ccccccc} & & & & b_1 & & b_0 \\ & & & & \uparrow & & \uparrow \\ 1 & \underbrace{1} & 1 & \underbrace{0} & \underbrace{0} & 01 & \underbrace{1} \\ & b_3 & & b_2 & & & \end{array}$$

**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.

$$\begin{array}{c} 0101 \\ \underbrace{\hspace{1cm}} \\ \text{Query} \end{array}$$

The sketch is

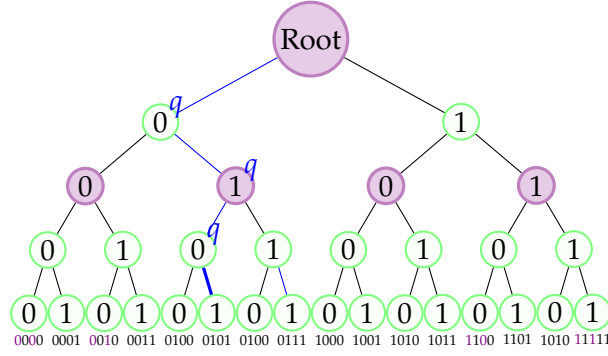


Figure 1.6: Example of Sketching

$$\begin{array}{cccc}
 \underbrace{0} & 0 & \underbrace{0} & 0 \\
 \underbrace{0} & 0 & \underbrace{1} & 0 \\
 \underbrace{1} & 1 & \underbrace{0} & 0 \\
 \underbrace{1} & 1 & \underbrace{1} & 1
 \end{array}$$

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  $\text{sketch}(x) \leq \text{sketch}(g) \leq \text{sketch}(x_{k+1})$ . 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  $\text{sketch}(e)$  among  $\text{sketch}(x_i)$ . Now we can find the predecessor  $\text{sketch}(e)$  among  $\text{sketch}(x_i)$ . Similarly we find the successor using the same algorithm.

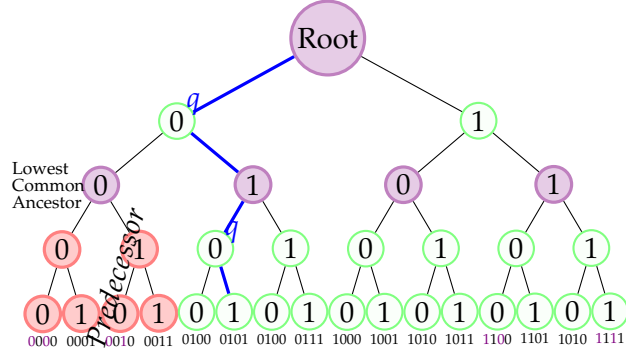


Figure 1.7: Example of Sketching

### Approximate sketch(x)

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

Idea : Mask important bits

$$x' = x \text{ and } (\sum_{i=1}^{r-1} 2^{b_i}) \text{ and multiple } x' \cdot m = (\sum_{i=1}^{r-1} x_{b_i} 2^{b_i}) ((\sum_{j=1}^{r-1} 2^{m_j}))$$

$$= (\sum_{i=1}^{r-1} \sum_{j=1}^{r-1} 2^{b_i+m_j})$$

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 for all  $i, j$
2.  $b_0 + m_0 < \dots < b_{r-1} + m_{r-1}$
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  $b_i + m'_j < \text{mod } r^3$

Property 1:

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

$$\forall \underbrace{i}_t, \underbrace{j}_r, \underbrace{k}_r$$

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

Property 2:

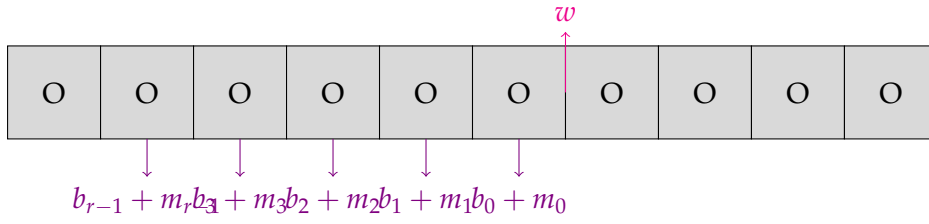


Figure 1.8: Description of Proof

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  $k$ th 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  $k$ th smallest element.

That is the focus of this lecture.

### Randomized Divide and Conquer

Rand-select ( $A - p - q - \underbrace{\quad}_i$ )  
ith smallest element in  $A$  to

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

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  $\tau$ , sequence of states/observations and actions  $D := (s_1, a_1, \dots, s_T)$
- reward function  $r(s, a)$
- Policy  $\pi(a, s)$  or  $\pi(a, o)$

$$\min_{\theta} \mathbb{E}_{\tau \sim P_{\cdot}(\theta)} \left[ \sum_t^T r(s_t, a_t) \right]$$

### 3.2 Imitation Learning

What is the goal of imitation learning ?

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

#### Imitation learning -Version 0

##### Deterministic policy

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



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

### Evaluation of the RL objective

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

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

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 the derivation of  $\log P_{\theta}(\tau)$  which is  $\frac{1}{P_{\theta}(\tau)}$

This means that  $\int \nabla_{\theta} P_{\theta}(\tau) r(\tau) d\tau$  is  $= \int \nabla_{\theta} \log P_{\theta}(\tau) r(\tau) d\tau$

We know that  $P_{\theta}(\tau) = P(s_t) \prod_t \pi_{\theta}(a_t, s_t) P(s_{t+1} | (s_t, a_t))$

So  $\log P_{\theta}(\tau) = \log P(s_t) + \sum_t \log \pi_{\theta}(a_t, s_t) + \log P(s_{t+1} | (s_t, a_t))$

$\log P(s_t)$  and  $\log P(s_{t+1} | (s_t, a_t))$  are constants.

$$\nabla \log P_{\theta}(\tau) = \nabla_{\theta} \sum_t \log \pi_{\theta}(a_t, s_t)$$

Expectation over trajectories sampled by or policy is  
 $\mathbb{E}_{\tau \sim P_{\theta}(\tau)}$

So the policy gradient now is  $\nabla_{\theta} J(\theta) = \mathbb{E}_{\tau \sim P_{\theta}(\tau)} \left[ \sum_t^T \nabla_{\theta} \log \pi(a_t, s_t) \right] \left[ \sum_t^T r(s_t, a_t) \right]$

The equation shown above is the gradient of the policy of the RL objective.

### Estimating the Gradient

This is the full algorithm.

1. Sample  $r_i$  from  $\pi_{\theta}(a_t | s_t)$
2.  $\nabla_{\theta} J(\theta)$  from  $\pi_{\theta}(a_t | s_t) \approx \sum_i (\sum_t \nabla_{\theta} \log \pi_{\theta}(a_t^i s_t^i)) (\sum_t (a_t^i s_t^i))$
3.  $\theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta)$
1. The equation in the first box is for imitation gradient, weighted by reward
2. The 2nd equation is for imitation learning

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_i^N \left( \sum_i^T \nabla_{\theta} \log \pi_{\theta}(a_{i,t} s_{i,t}) \right) \left( \sum_i^T r(s_{i,t}, a_{i,t}) \right)$$

$$\nabla_{\theta} J_{BC}(\theta) \approx \frac{1}{N} \sum_i^N \left( \sum_i^T \nabla_{\theta} \log \pi_{\theta}(a_{i,t} s_{i,t}) \right)$$

Figure 3.1: Imitation gradient and learning

This increases the likelihood of action when the rewards are high and decreases the likelihood of actions in trajectories with low reward.

## Off-policy version of Policy Gradient

Importance Sampling - Using proposal distribution  $q$   $\mathbb{E}_{x \sim P(x)}[f(x)] = \int P(x)f(x)dx = \int \frac{q(x)}{q(x)}P(x)f(x)dx = \int q(x)\frac{p(x)}{q(x)}f(x)dx = \mathbb{E}_{x \sim q(x)} \left[ \frac{p(x)}{q(x)} \right] f(x)$  Please note: Important for  $q$  to have non-zero support for high probability  $P(X)$

Figure 3.2: Off-policy Gradient

$q(x)$  is considered the old policy.  $\mathbb{E}_{x \sim q(x)} \left[ \frac{p(x)}{q(x)} \right] f(x)$  and we are sampling from the old policy and weighting by dividing the new policy by the old policy.

**What is we want to use samples from  $\bar{P}(\tau)$  ?**

$$J(\theta) = \mathbb{E}_{\tau \sim \bar{P}_\theta(\tau)} \left[ \frac{P_\theta(\tau)}{\bar{P}(\tau)} \right] r(\tau)$$