Succinct Data Structures

Lecture Notes

1 Overview

Topics covered:

- Survey of succinct data structures
- Succinct binary tree (level order representation)
- Via balanced parentheses
- Succinct Rank & Select

Goal: "Small" space (often static)
Easy to do in linear space. But linear space is not optimal!

1.1 Three Senses of Small Space

1. **Implicit:** $\text{OPT} + O(1)$ bits
   - OPT = information theoretic optimum
   - $O(1)$ is for rounding
   - Typically, the data structure is "just the data" permuted in some order
   - Examples: sorted array, heap
2. **Succinct:** $\text{OPT} + o(\text{OPT})$ bits
   - Leading constant is 1
3. **Compact:** $O(\text{OPT})$ bits
   - Often a factor w smaller than "linear space" data structures
   - "Linear space" DS use $O(n)$ words for n-bit strings
   - Example: Suffix trees use $O(n)$ words for n-bit strings

2 Mini Survey

- **Implicit dynamic search tree:** $O(\lg n)$ worst case insert/delete/predecessor
- **Succinct dictionary:**
$$\lg \binom{u}{n} + O\left(n \cdot \frac{(\lg \lg n)^2}{\lg n}\right) \text{ bits}$$
- **Succinct binary trie:**
$$C_n = \binom{2n}{n} \frac{1}{n+1} \sim 4^n \text{ such tries}$$
$$\lg C_n + o(\lg C_n) = 2n + o(n) \text{ bits}$$
- **Succinct k-ary trie:**
$$C_k = \binom{kn+1}{n} \frac{1}{kn+1} \text{ tries}$$

3 Level Order Representation of Binary Tries

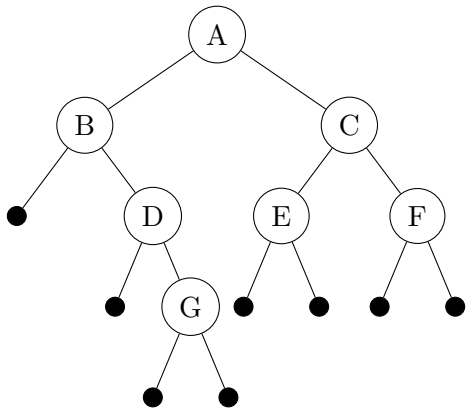
For each node in level order:

- Write 0/1 for whether it has a left child
- Write 0/1 for whether it has a right child

$\Rightarrow 2n$ bits

3.1 Example

Consider the following binary trie:



Level order traversal: A, B, C, D, E, F, G
Bit string (with external nodes, $2n+1$ bits):

Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Bit	(1)	1	1	0	1	1	1	0	1	0	0	0	0	0	0
Node	A	B	C	•	D	E	F	•	G	•	•	•	•	•	•

The first bit (1) in parentheses represents the root node A. Internal nodes are marked with 1, external nodes (•) with 0.

$B = 111011101000000$

Equivalent formulation:

- Append external node • for each missing child
- For each node in level order: write 0 if external, 1 if internal
- $\Rightarrow$ extra leading 1 $(2n + 1)$ bits

4 Navigation

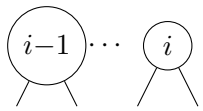
Theorem 1. *Left and right children of the i-th internal node are at positions $2i$ and $2i + 1$ in the array.*

Proof. By induction on i:

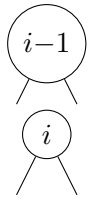
The children of node i appear just after $(i - 1)$ internal nodes' children, as external nodes have no children.

Consider two cases: either node i is on the same level as node $i - 1$, or on a new level.

Case 1:



Case 2:



Before node i's children, we have:

- $i - 1$ internal nodes
- j external nodes

Remaining children of internal nodes before position of i's left child:

$$= 2(i - 1) - (i - 1) - j = i - j - 1$$

where $2(i - 1)$ counts all children, $(i - 1)$ subtracts internal nodes already seen, and j subtracts external nodes.

Position of left child: $= i - j - 1 + (i + j) + 1 = 2i$

Position of right child: $= 2i + 1$ □

5 Rank-Select in Bit Strings

Definition 1. *For a bit string B:*

- *$rank_1(i)$ = number of 1s at or before position i*
- *$select_1(j)$ = position of the j-th 1 bit*

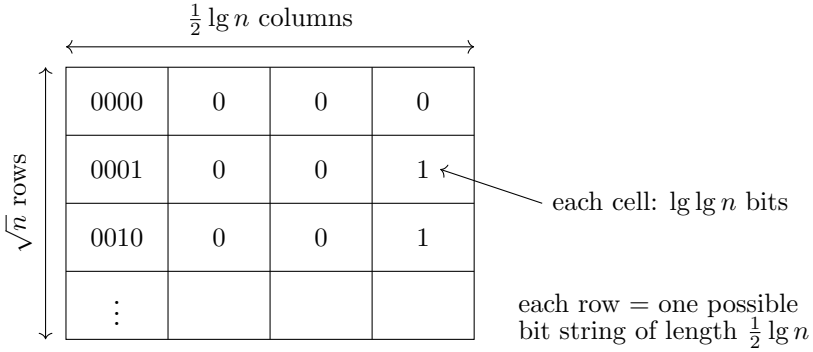
Navigation using Rank-Select:

$$\begin{aligned} \text{left\_child}(i) &= 2 \cdot \text{rank}_1(i) \\ \text{right\_child}(i) &= 2 \cdot \text{rank}_1(i) + 1 \\ \text{parent}(i) &= \text{select}_1(\lfloor i/2 \rfloor) \end{aligned}$$

Note: Subtree size is *not* possible in level order representation.

6 Rank: Jacobson (FOCS 1989)

Step 1: Use lookup table for substrings of length $\frac{1}{2}\lg n$
The lookup table has the following structure:



- **Rows:** $2^{\frac{1}{2}\lg n} = \sqrt{n}$ possible bit strings of length $\frac{1}{2}\lg n$
- **Columns:** $\frac{1}{2}\lg n$ possible query positions within each bit string
- **Cell size:** Each answer is at most $\frac{1}{2}\lg n$, requiring $\lg(\frac{1}{2}\lg n) = O(\lg\lg n)$ bits

Total space for lookup table:

$$\underbrace{\sqrt{n}}_{\#\text{ rows}} \times \underbrace{\frac{1}{2}\lg n}_{\#\text{ columns}} \times \underbrace{O(\lg\lg n)}_{\text{bits per cell}} = O(\sqrt{n} \cdot \lg n \cdot \lg\lg n) = o(n) \text{ bits}$$

Step 2: Split into $(\lg^2 n)$-bit chunks

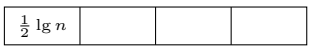
Store cumulative rank: $\lg n$ bits

$\lg^2 n$			

Space:

$$O\left(\frac{n}{\lg^2 n} \cdot \lg n\right) = O\left(\frac{n}{\lg n}\right) \text{ bits}$$

Step 3: Split each chunk into $\frac{1}{2}\lg n$-bit subchunks



Store cumulative rank within chunk: $\lg\lg n$ bits

$\lg^2 n$ total

Space:

$$O\left(\frac{n}{\lg n} \cdot \lg\lg n\right) = o(n) \text{ bits}$$

Step 4: Query

$$\text{rank}(i) = \underbrace{\text{rank of chunk}}_{\text{Step 2}} + \underbrace{\text{relative rank of subchunk within chunk}}_{\text{Step 3}} + \underbrace{\text{relative rank of element within subchunk}}_{\text{lookup table}}$$

Result: $O(1)$ time, $O\left(n \cdot \frac{\lg\lg n}{\lg n}\right)$ bits
$\Rightarrow O\left(\frac{n}{\lg^k n}\right)$ bits possible for any $k = O(1)$
$\Rightarrow O\left(\frac{\lg n}{\lg\lg n}\right)$ insert/delete/rank/select

7 Select: Clark-Munro (1996)

Step 1: Store array of indices of every $(\lg n \cdot \lg\lg n)$-th 1 bit
Space:

$$O\left(\frac{n}{\lg n \cdot \lg\lg n} \cdot \lg n\right) = O\left(\frac{n}{\lg\lg n}\right) \text{ bits}$$

Step 2: Within group of $\lg n \cdot \lg\lg n$ 1-bits, say r bits total.
If $r \geq (\lg n \cdot \lg\lg n)^2$, then store array of indices of 1 bits in group.
Space:

$$O\left(\frac{n}{(\lg n \cdot \lg\lg n)^2} \cdot \lg n \cdot \lg\lg n \cdot \lg n\right) = O\left(\frac{n}{\lg\lg n}\right) \text{ bits}$$

$$\underbrace{\frac{n}{(\lg n \cdot \lg\lg n)^2}}_{\#\text{ such groups}} \times \underbrace{\lg n \cdot \lg\lg n}_{\#\text{ 1 bits}} \times \underbrace{\lg n}_{\text{index size}}$$

Step 3: Repeat (1) & (2) on all reduced bit strings to reduce to bit strings of length $(\lg\lg n)^{O(1)}$

(3.1) Store relative index (using $\lg\lg n$ bits) of $(\lg\lg n)^2$-th 1 bit
Space:

$$O\left(\frac{n}{(\lg\lg n)^2} \cdot \lg\lg n\right) = O\left(\frac{n}{\lg\lg n}\right) \text{ bits}$$

(3.2) Within group of $(\lg\lg n)^2$ 1-bits, say r bits.

If $r \geq (\lg\lg n)^4$, then store relative indices of 1 bits.

Space:

$$O\left(\frac{n}{(\lg\lg n)^4} \cdot (\lg\lg n)^2 \cdot \lg\lg n\right) = O\left(\frac{n}{\lg\lg n}\right) \text{ bits}$$

Else reduce to bit strings of length $r \leq (\lg\lg n)^4$.

Step 4: Use lookup table for bit strings of length $\leq \frac{1}{2}\lg n$
Space:

$$O(\sqrt{n} \cdot \lg n \cdot \lg\lg n)$$

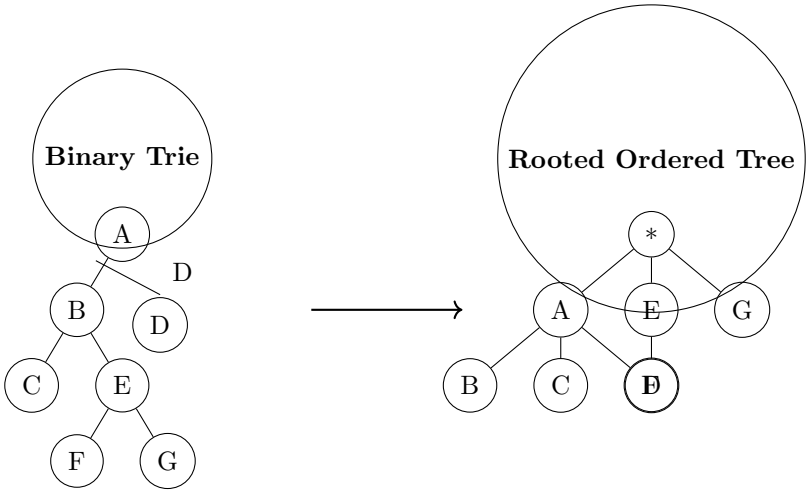
$$\underbrace{\sqrt{n}}_{\#\text{ bit strings}} \quad \times \quad \underbrace{\lg n}_{\#\text{ queries}} \quad \times \quad \underbrace{\lg\lg n}_{\text{answer size}}$$

Result: $O(1)$ query, $O\left(\frac{n}{\lg\lg n}\right)$ bits
$\Rightarrow O\left(\frac{n}{\lg^k n}\right)$ bits for any $k = O(1)$

8 Binary Trees as Balanced Parentheses

Reference: Munro & Raman 2001

8.1 Bijection: Binary Trie ↔ Rooted Ordered Tree



Query correspondence:

Binary Trie	$\longrightarrow$	**Rooted Ordered Tree**
node	$\rightarrow$	node
left child	$\rightarrow$	first child
right child	$\rightarrow$	next sibling
parent	$\rightarrow$	prev. sibling or parent

Subtree size:

$$\text{subtree\_size(binary trie)} = \text{size(node)} + \text{size(right sibling)}$$

8.2 Balanced Parentheses

DFS traversal with open parenthesis on first visit, close parenthesis on second visit (Euler tour).

Example:

```
( ( ( ) ( ) ( ) ) ( ( ) ) ( ) )
* A B B C C D D A E F F E G G *
```

Bit encoding:

- Open parenthesis = 0
- Closed parenthesis = 1

Subtree size:

$$= \frac{1}{2} \times \text{distance to matching closing parenthesis}$$

9 References

1. G. Jacobson. *Succinct Static Data Structures.* PhD Thesis, Carnegie Mellon University, 1989.
2. D. Clark and J. I. Munro. *Efficient Suffix Trees on Secondary Storage.* SODA, 383–391, 1996.
3. J. I. Munro and V. Raman. *Succinct Representation of Balanced Parentheses and Static Trees.* SIAM J. Computing, 31(3):762–776, 2001.