

RB-Tree Augmentation

- Augment x with **Size(x)**, where
 - $\text{Size}(x)$ = size of subtree rooted at x
 - $\text{Size}(\text{NIL}) = 0$

Augmented Data Structures

- Why is it needed?
 - Because basic data structures not enough for all operations
 - storing extra information helps execute special operations more efficiently.
- Can any data structure be augmented?
 - **Yes**. Any data structure can be augmented.
- Can a data structure be augmented with any additional information?
 - Theoretically, **yes**.
- How to choose which additional information to store.
 - Only if we can **maintain** the additional information efficiently under all operations. That means, with additional information, we need to perform old and new operations efficiently maintain the additional information efficiently.

How to augment data structures

1. choose an underlying data structure
2. determine additional information to be maintained in the underlying data structure,
3. develop new operations,
4. verify that the additional information can be maintained for the modifying operations on the underlying data structure.

Augmenting RB-Trees

Theorem 14.1, page 309

Let f be a field that augments a red-black tree T with n nodes, and $f(x)$ can be computed using only the information in nodes x , $\text{left}[x]$, and $\text{right}[x]$, including $f[\text{left}[x]]$ and $f[\text{right}[x]]$.

Then, we can maintain $f(x)$ during insertion and deletion without asymptotically affecting the $O(\log n)$ performance of these operations.

For example,

$$\text{size}[x] = \text{size}[\text{left}[x]] + \text{size}[\text{right}[x]] + 1$$

$$\text{rank}[x] = ?$$

Examples of augmenting information for RB-Trees

- Parent
- Height
- Any associative function on all previous values or all succeeding values.
- Next
- Previous

Example

- $[0,6]$, $[1,4]$, $[2,13]$, $[3,5]$, $[3,8]$, $[5,7]$, $[5,9]$, $[6,10]$, $[8,11]$, $[8,12]$, $[12,14]$
- **Simple Greedy Selection**
 - Sort by start time and pick in "greedy" fashion
 - Does not work. WHY?
 - $[0,6]$, $[6,10]$ is the solution you will end up with.
- **Other greedy strategies**
 - Sort by length of interval
 - Does not work. WHY?

Example

- [0,6], [1,4], [2,13], [3,5], [3,8], [5,7], [5,9], [6,10], [8,11], [8,12], [12,14]
- [1,4], [3,5], [0,6], [5,7], [3,8], [5,9], [6,10], [8,11], [8,12], [2,13], [12,14] -- Sorted by finish times
- [1,4], [3,5], [0,6], [5,7], [3,8], [5,9], [6,10], [8,11], [8,12], [2,13], [12,14]
- [1,4], [3,5], [0,6], [5,7], [3,8], [5,9], [6,10], [8,11], [8,12], [2,13], [12,14]
- [1,4], [3,5], [0,6], [5,7], [3,8], [5,9], [6,10], [8,11], [8,12], [2,13], [12,14]
- [1,4], [3,5], [0,6], [5,7], [3,8], [5,9], [6,10], [8,11], [8,12], [2,13], [12,14]
- [1,4], [3,5], [0,6], [5,7], [3,8], [5,9], [6,10], [8,11], [8,12], [2,13], [12,14]

Greedy Algorithms

- Given a set of activities (s_i, f_i) , we want to schedule the maximum number of non-overlapping activities.
- GREEDY-ACTIVITY-SELECTOR (s, f)
 1. $n = \text{length}[s]$
 2. $S = \{a_1\}$
 3. $i = 1$
 4. **for** $m = 2$ **to** n **do**
 5. **if** s_m is not before f_i **then**
 6. $S = S \cup \{a_m\}$
 7. $i = m$
 8. **return** S

Why does it work?

- **THEOREM**

Let A be a set of activities and let a_1 be the activity with the earliest finish time. Then activity a_1 is in some maximum-sized subset of non-overlapping activities.

- **PROOF**

Let S' be a solution that does not contain a_1 . Let a'_1 be the activity with the earliest finish time in S' . Then replacing a'_1 by a_1 gives a solution S of the same size.

Why are we allowed to replace? Why is it of the same size?

Then apply induction! **How?**

Greedy Algorithms – Huffman Coding

- Huffman Coding Problem

Example: Release 29.1 of 15-Feb-2005 of [TrEMBL](#) Protein Database contains **1,614,107** sequence entries, comprising **505,947,503** amino acids. There are 20 possible amino acids. What is the minimum number of bits to store the compressed database?

~2.5 G bits or 300MB.

- How to improve this?

- Information: **Frequencies are not the same.**

Ala (A) 7.72	Gln (Q) 3.91	Leu (L) 9.56	Ser (S) 6.98
Arg (R) 5.24	Glu (E) 6.54	Lys (K) 5.96	Thr (T) 5.52
Asn (N) 4.28	Gly (G) 6.90	Met (M) 2.36	Trp (W) 1.18
Asp (D) 5.28	His (H) 2.26	Phe (F) 4.06	Tyr (Y) 3.13
Cys (C) 1.60	Ile (I) 5.88	Pro (P) 4.87	Val (V) 6.66

- **Idea:** Use shorter codes for more frequent amino acids and longer codes for less frequent ones.

Huffman Coding

2 million characters in file.

A, C, G, T, N, Y, R, S, M

IDEA 1: Use ASCII Code

Each need at least 8 bits,
Total = 16 M bits = **2 MB**

IDEA 2: Use 4-bit Codes

Each need at least 4 bits,
Total = 8 M bits = **1 MB**

IDEA 3: Use Variable Length

Codes

A	22	11
T	22	10
C	18	011
G	18	010
N	10	001
Y	5	00011
R	4	00010
S	4	00001
M	3	00000

How to Decode?

Need Unique decoding!
Easy for Ideas 1 & 2.
What about Idea 3?

110101101110010001100000000110

110101101110010001100000000110

Percentage
Frequencies

2 million characters in file.

Length = ?

Expected length = ?

Sum up products of frequency times the code length, i.e.,

$$(.22 \times 2 + .22 \times 2 + .18 \times 3 + .18 \times 3 + .10 \times 3 + .05 \times 5 + .04 \times 5 + .04 \times 5 + .03 \times 5) \times 2 \text{ M bits} =$$

$$3.24 \text{ M bits} = \mathbf{.4 \text{ MB}}$$