Fenwick Trees

By Mark Galloway

Overview

- Motivation
- The Fenwick Tree
- Applications
- Example
- Source Code
- Time Complexity

Motivation: Cumulative Frequencies

- We want to store and query a range of frequencies
  - eg/ test scores of 11 students
  - range [1, 10]

- We want to perform Range Sum Query (RSQ) operations on this data
  - RSQ(1.. 10) = the cumulative frequency of all scores
  - RSQ(1..5) = the cumulative frequency of scores <= 5
  - RSQ(8..10) = the number of A’s received

Motivation: Cumulative Frequencies

- Naive Implementation: An array
  - m = [2, 4, 5, 5, 6, 6, 6, 7, 7, 8, 9]
  - f = [0, 1, 1, 2, 4, 7, 9, 10, 11, 11]

RSQ(1) = 0
RSQ(1..5) = 4

- Runtime Analysis
  - This is O(N) per query. We can do better.

Formal Definition

A Fenwick tree or binary indexed tree is a data structure providing efficient methods for calculation and manipulation of the cumulative sums of a table of values.

Fenwick Trees provide a method to query the running total at any index, in addition to allowing changes to the underlying value table and having all further queries reflect those changes.

History

- Proposed by Peter Fenwick in 1994
  - University of Auckland, New Zealand

- Also called Binary Indexed Tree (BIT)

- Originally designed for dynamic arithmetic data compression

BIT Applications

- Arithmetic Coding
  - Used in lossless data compression

- Dynamic Cumulative Frequency Tables
  - Range Sum Query

Implementation

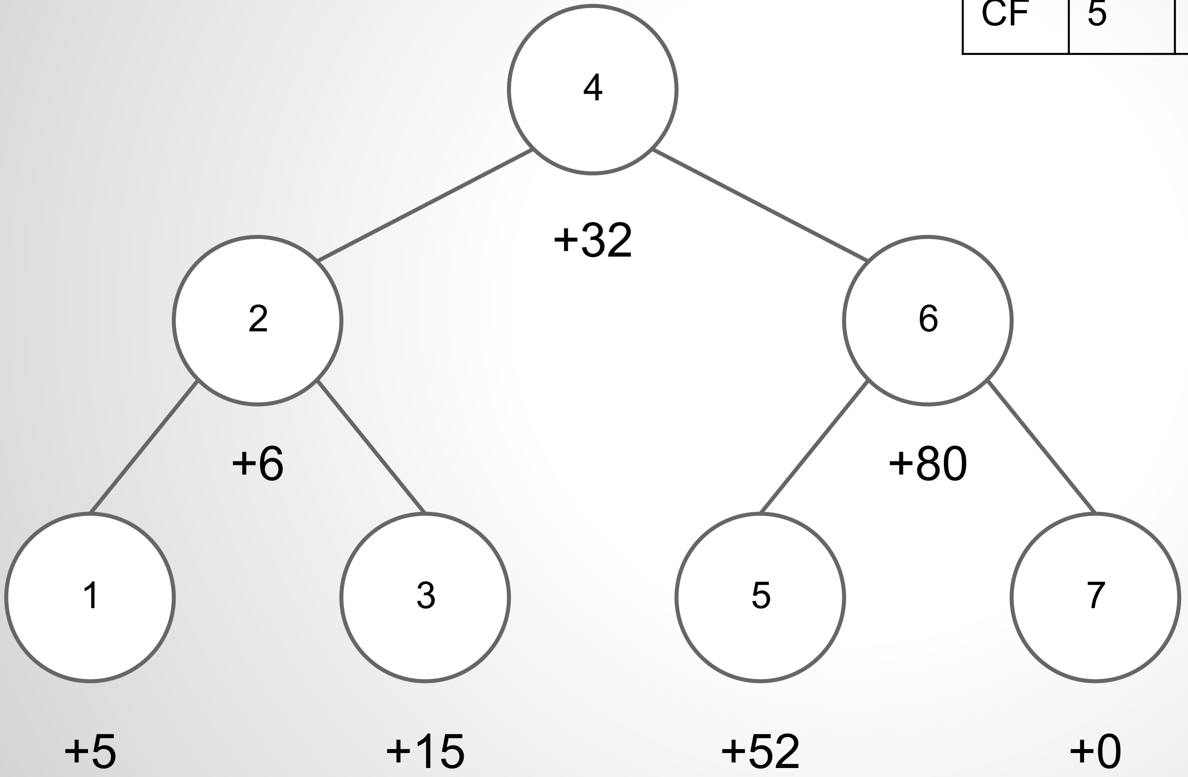
- Implemented as a flat array

- Each index contains a pre-calculated sum of a subsection of the table. Combining sums in an upward traversal to the root provides the desired range sum.

- The index of a vertex's parent or child is calculated through bitwise operations on the binary representation of its index

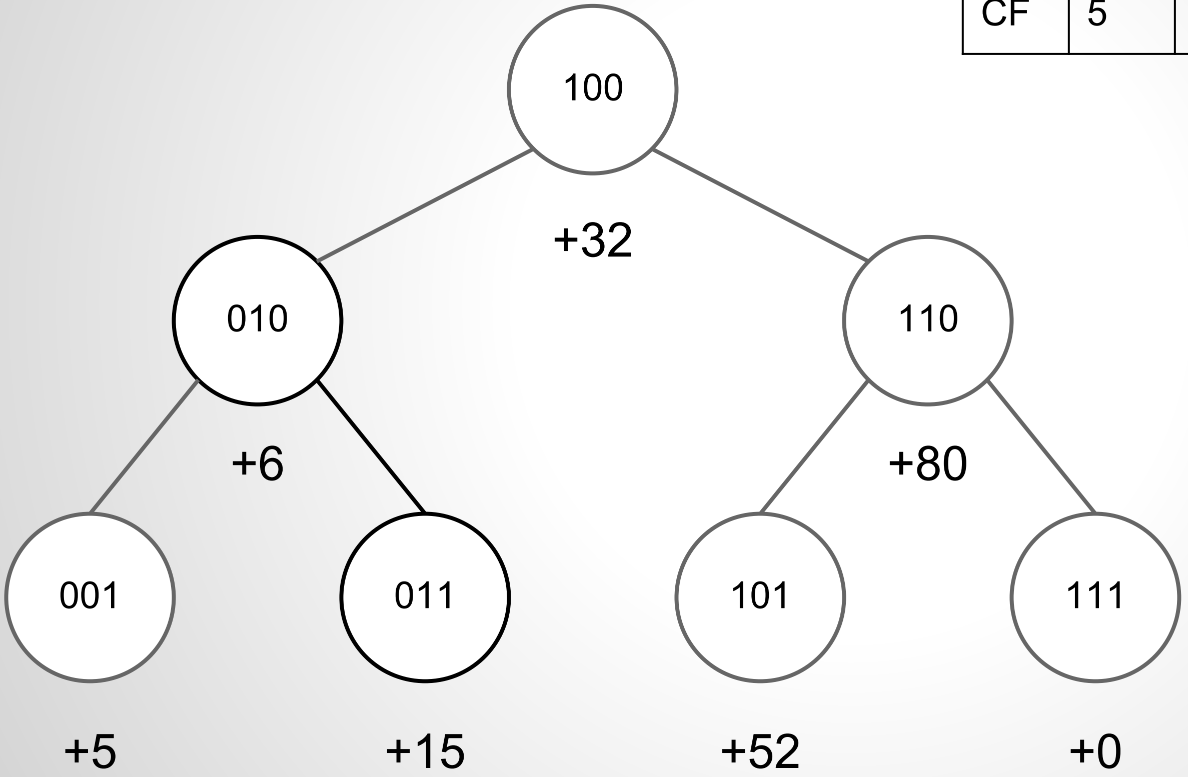
Example

N	1	2	3	4	5	6	7
F	5	1	15	11	52	28	0
CF	5	6	21	32	84	112	112



Example

N	1	2	3	4	5	6	7
F	5	1	15	11	52	28	0
CF	5	6	21	32	84	112	112



BIT Operations: rsq(b)

Let LSB = Least Significant Bit

Let b be an index <= N

Let b’ = b - LSB(b)

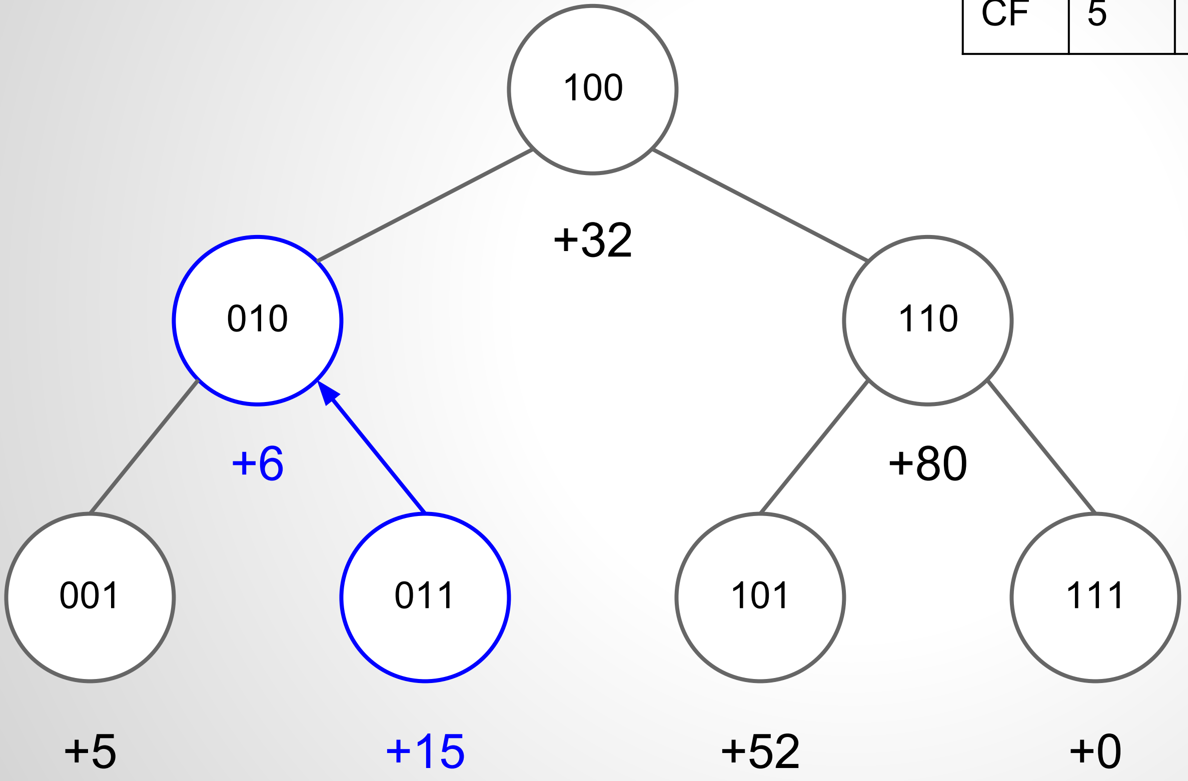
rsq(b) = sum(b + b’ + b’’ + b’’’+..... + b^i)

where b^i is 0

rsq(a, b) = rsq(b) - rsq(a)

Example: rsq

N	1	2	3	4	5	6	7
F	5	1	15	11	52	28	0
CF	5	6	21	32	84	112	112

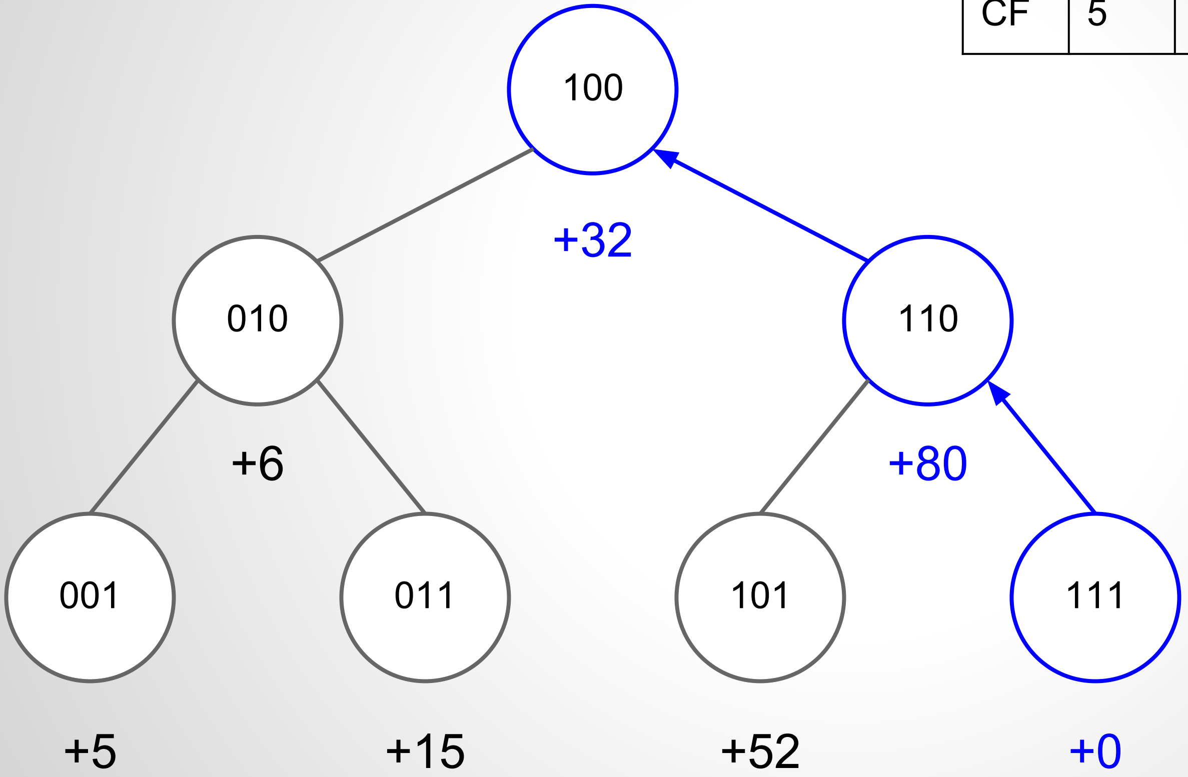


eg/ rsq(3)

= rsq(011)
= sum(011 + 010 + 000)
= 15 + 6
= 21

Example: rsq

N	1	2	3	4	5	6	7
F	5	1	15	11	52	28	0
CF	5	6	21	32	84	112	112



eg/ rsq(7)

= rsq(111)
= sum(111 + 110+100 + 000)
= 0 + 80 + 32
= 112

BIT Operations: update(b,v)

Let LSB = Least Significant Bit

Let b be an index <= N

Let b’ = b + LSB(b)

Let v = a frequency increment/decrement

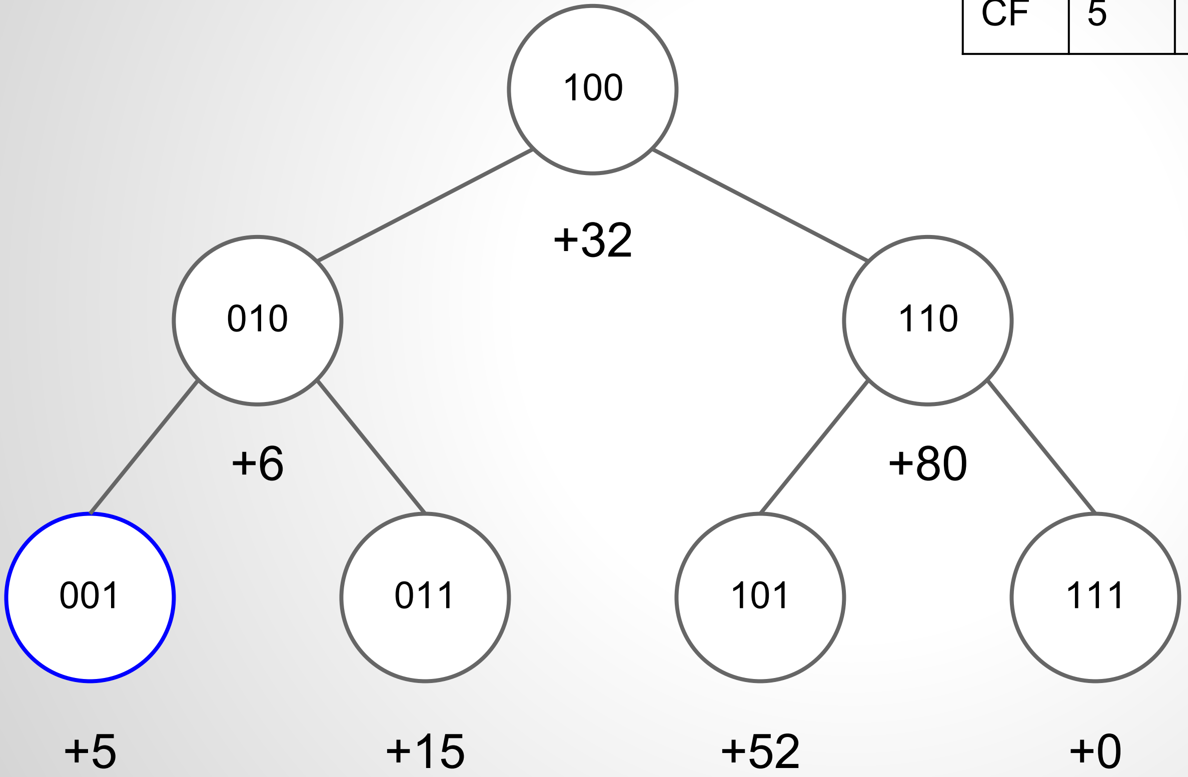
update(b, v) = update(b, v), update(b’, v) … update(b^i, v)

where b^i + LSB(b’) would exceed N.

Example: update

N	1	2	3	4	5	6	7
F	5	1	15	11	52	28	0
CF	5	6	21	32	84	112	112

eg/ update(1,5)

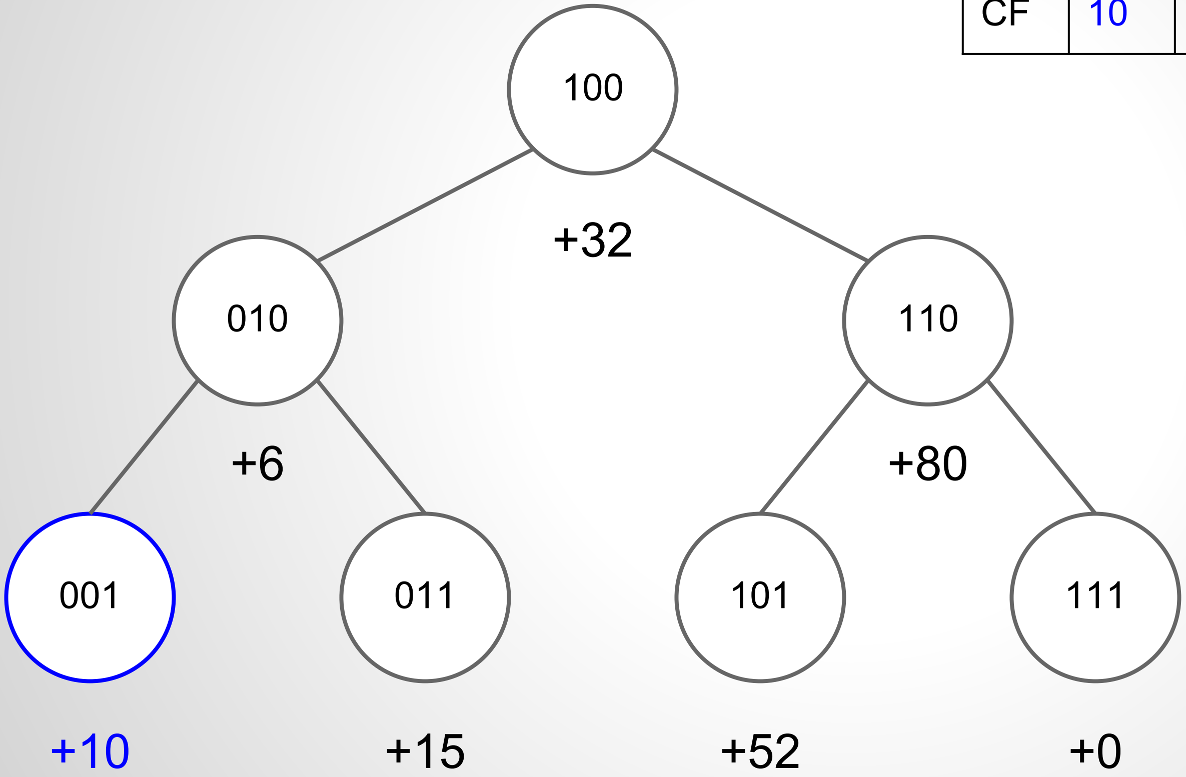


Example: update

N	1	2	3	4	5	6	7
F	10	1	15	11	52	28	0
CF	10	6	21	32	84	112	112

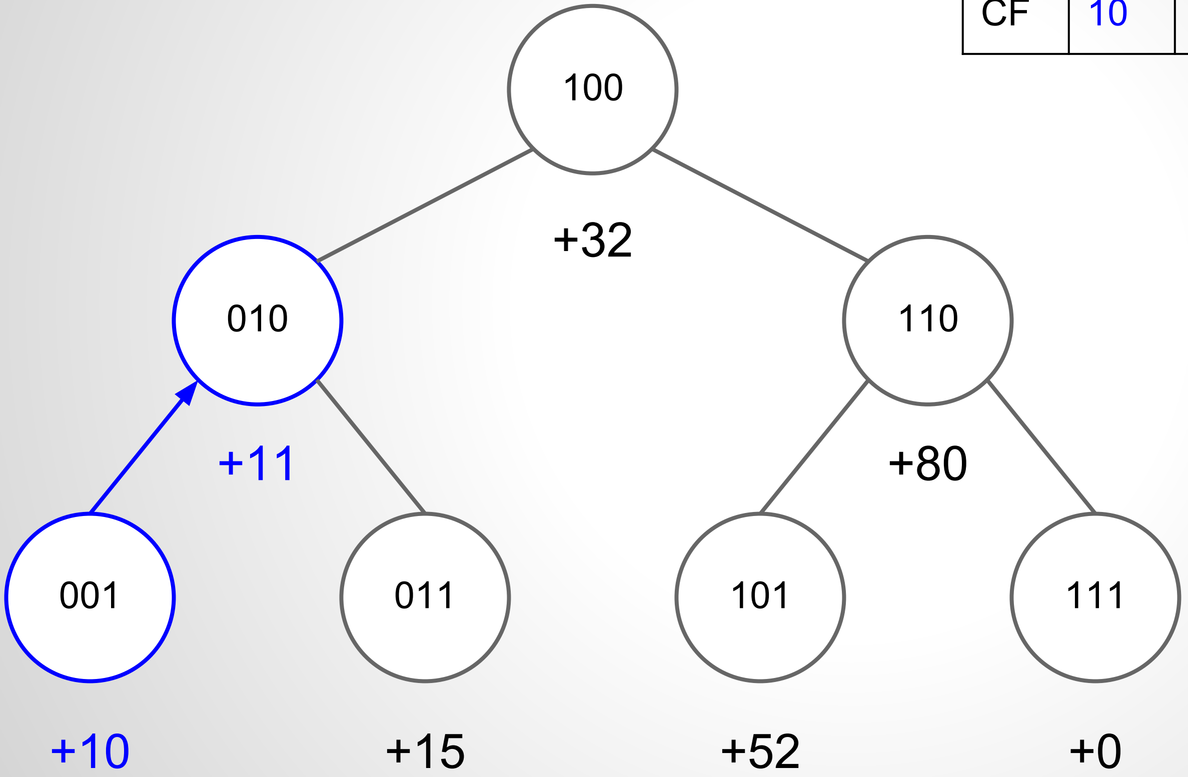
eg/ update(1,5)

= update(001, 5)



Example: update

N	1	2	3	4	5	6	7
F	10	1	15	11	52	28	0
CF	10	11	26	32	84	112	112

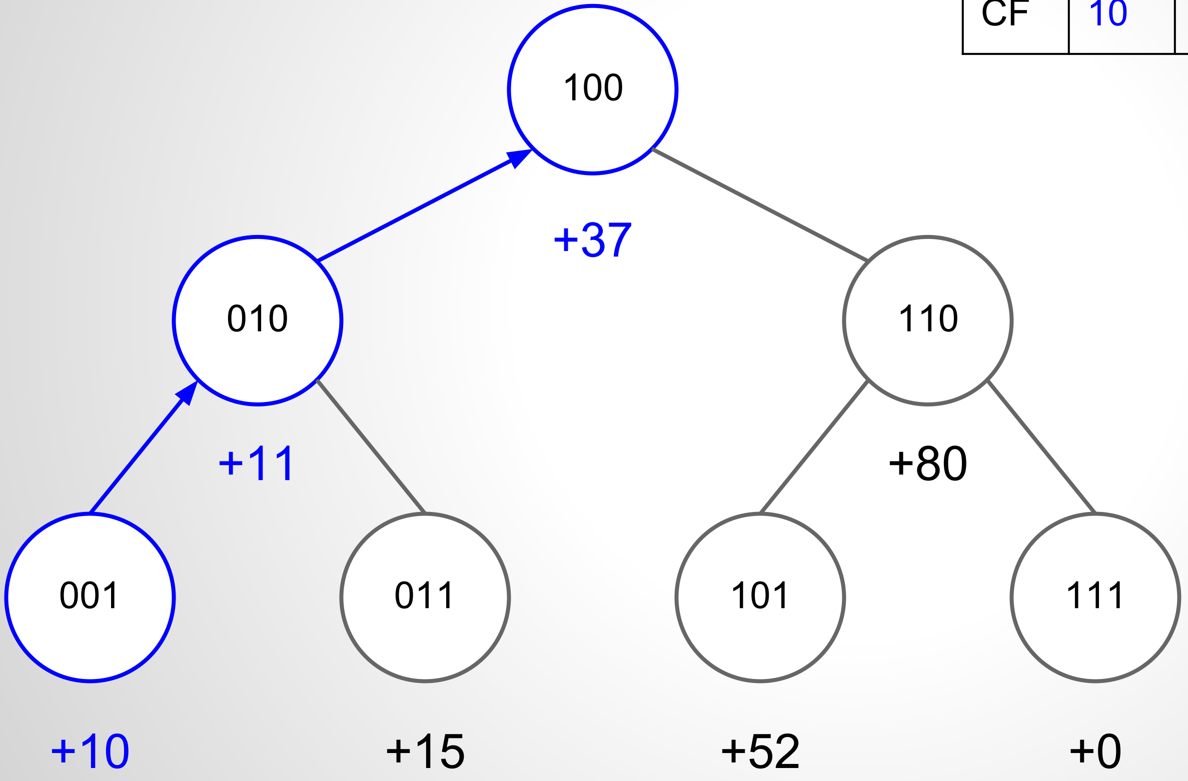


eg/ update(1,5)

= update(001, 5),
update(010, 5)

Example: update

N	1	2	3	4	5	6	7
F	10	1	15	11	52	28	0
CF	10	11	26	37	89	117	117



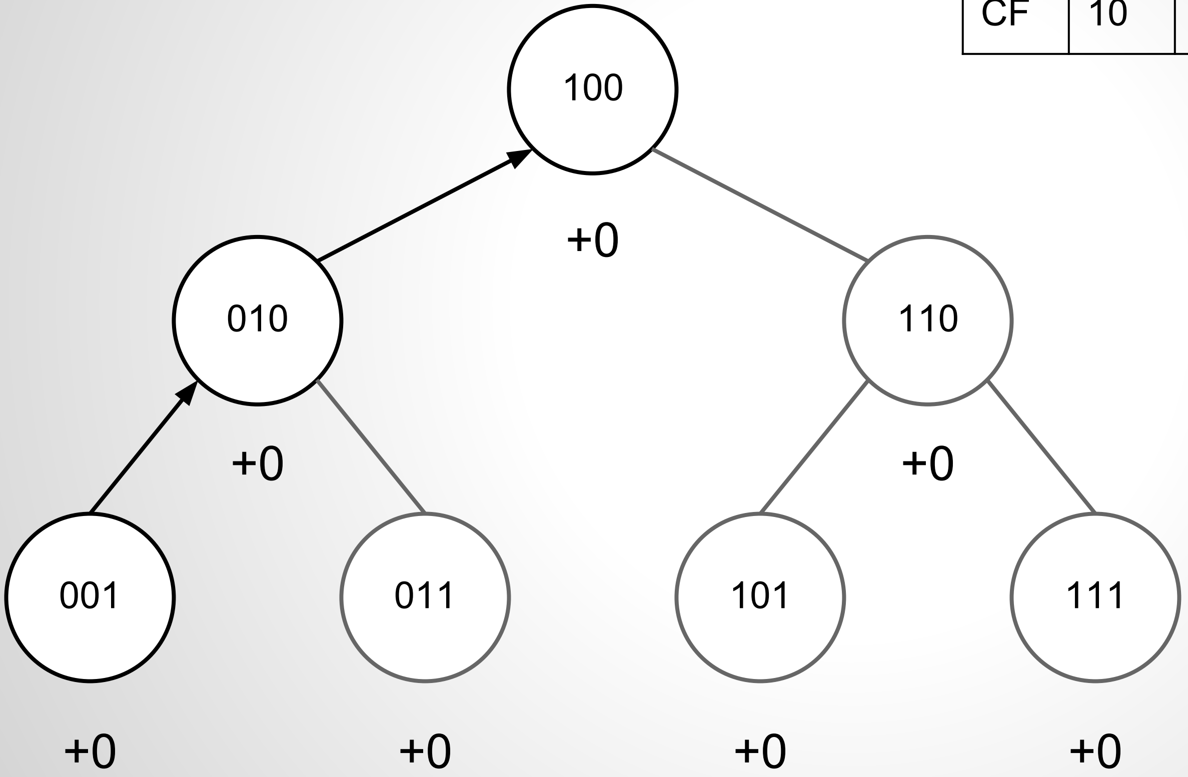
eg/ update(1,5)

= update(001, 5),
update(010, 5),
update(100, 5)

1000 > N, so we have reached the root

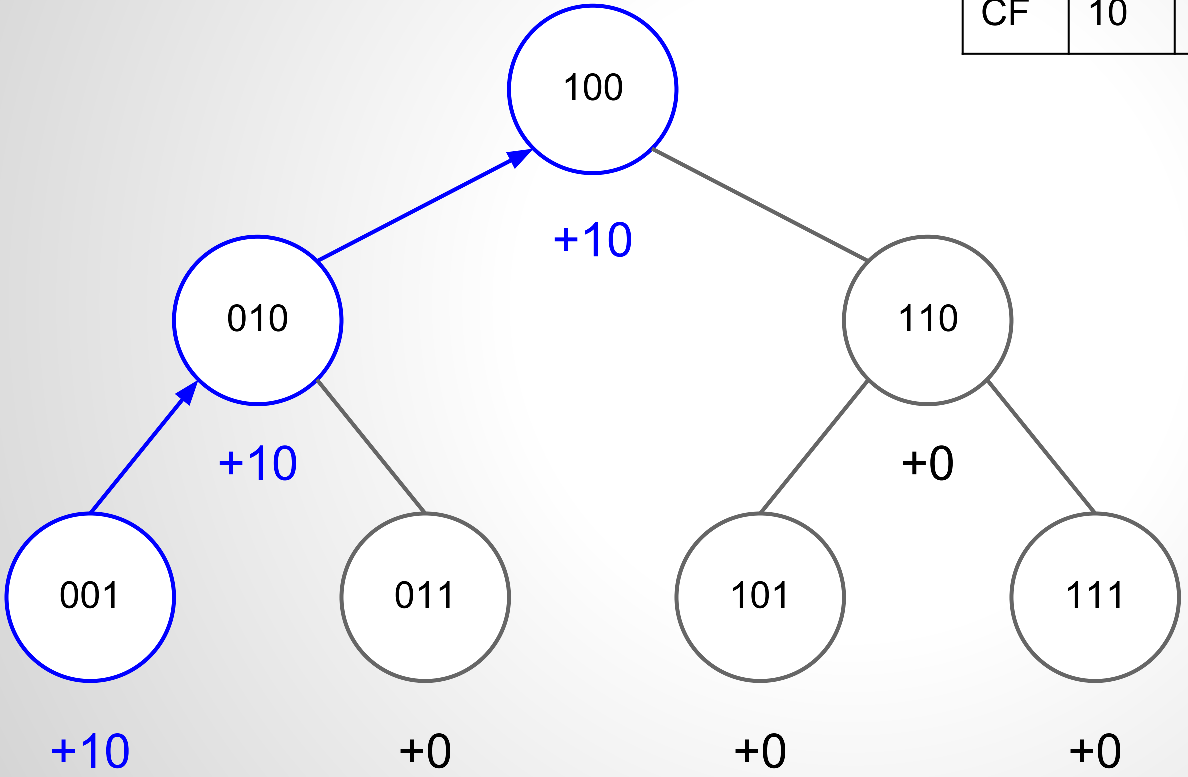
Example: create

N	1	2	3	4	5	6	7
F	10	1	15	11	52	28	0
CF	10	11	26	37	89	117	117



Example: create

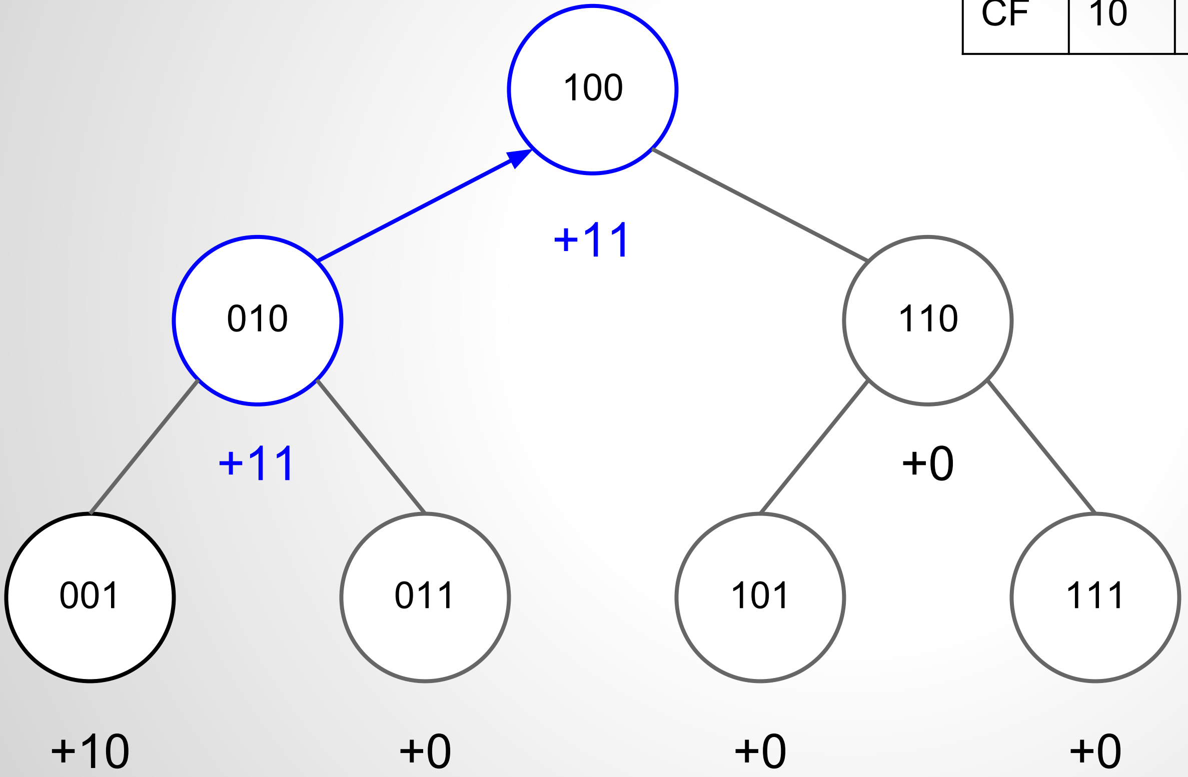
N	1	2	3	4	5	6	7
F	10	1	15	11	52	28	0
CF	10	11	26	37	89	117	117



update(1, 10)

Example: create

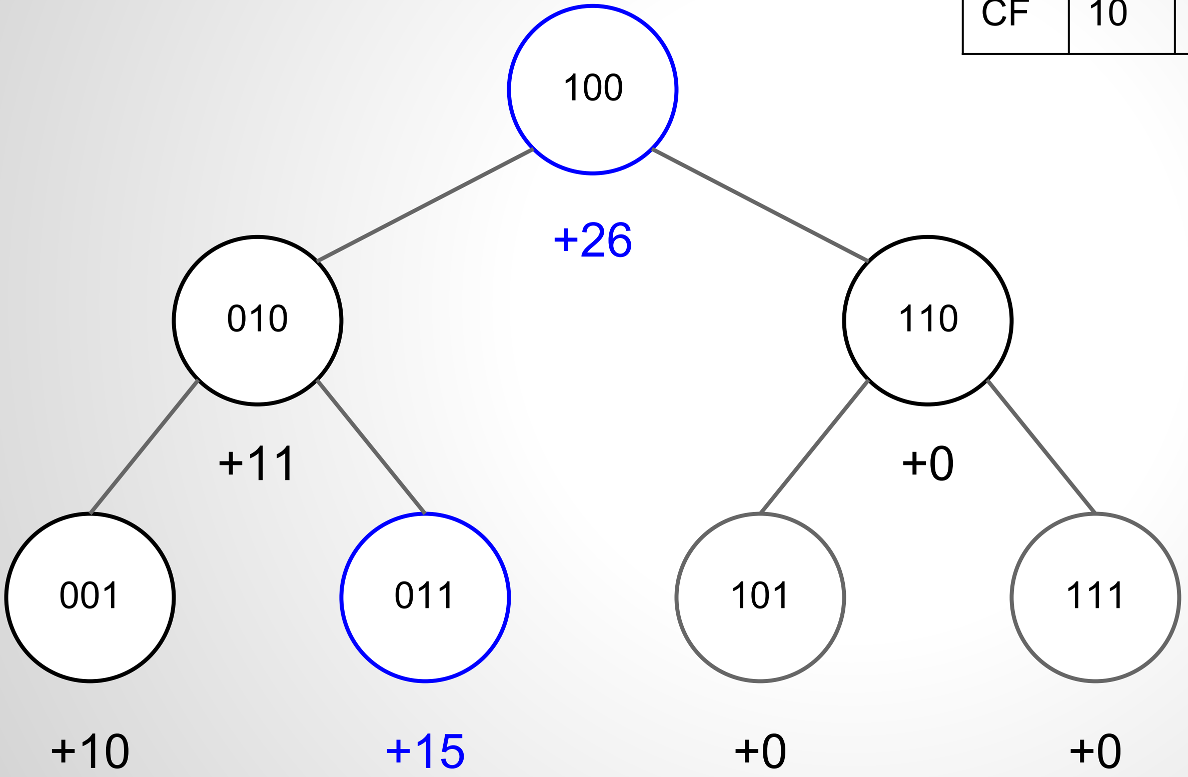
N	1	2	3	4	5	6	7
F	10	1	15	11	52	28	0
CF	10	11	26	37	89	117	117



update(1, 10)
update(2,   1)

Example: create

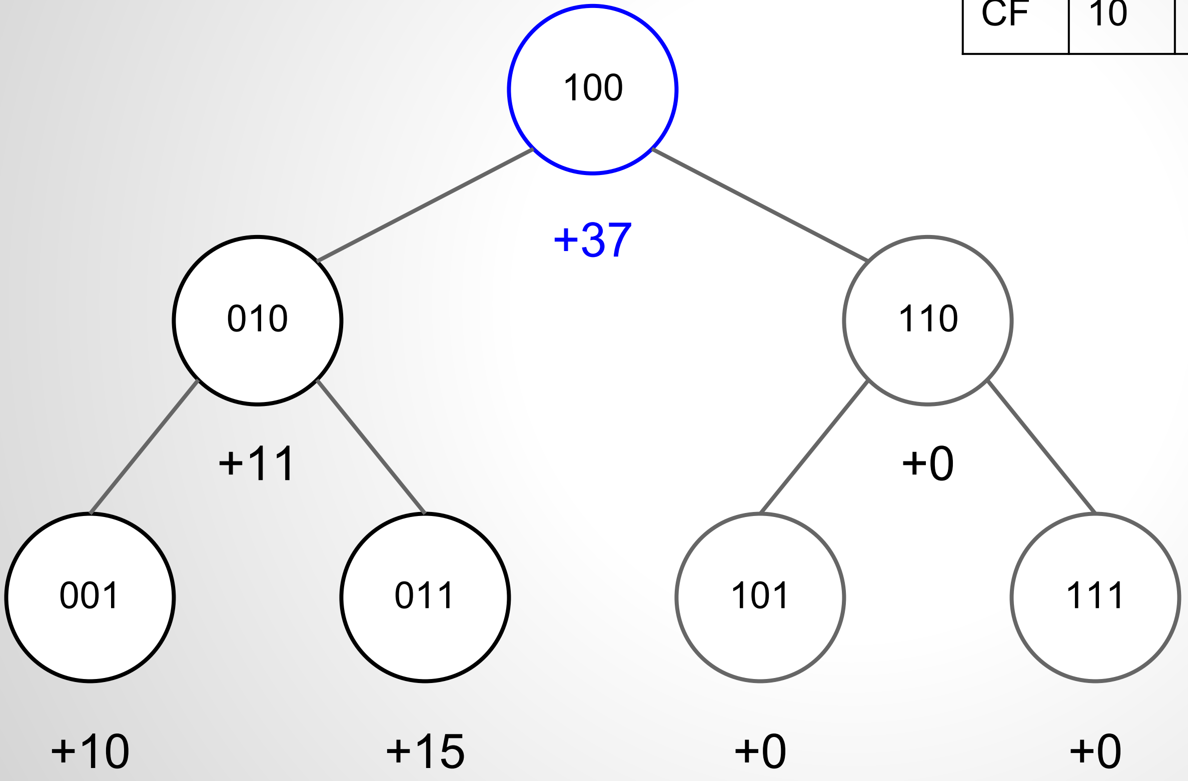
N	1	2	3	4	5	6	7
F	10	1	15	11	52	28	0
CF	10	11	26	37	89	117	117



update(1, 10)
update(2,   1)
update(3, 15)

Example: create

N	1	2	3	4	5	6	7
F	10	1	15	11	52	28	0
CF	10	11	26	37	89	117	117



update(1, 10)
update(2, 1)
update(3, 15)
update(4, 11)

Source Code

```
#define LSOne(S) (S & (-S))

class FenwickTree {
private:
  vi ft;

public:
  FenwickTree() {}
  // initialization: n + 1 zeroes, ignore index 0
  FenwickTree(int n) { ft.assign(n + 1, 0); }

  int rsq(int b) {                                     // returns RSQ(1, b)
    int sum = 0; for (; b; b -= LSOne(b)) sum += ft[b];
    return sum; }

  int rsq(int a, int b) {                              // returns RSQ(a, b)
    return rsq(b) - (a == 1 ? 0 : rsq(a - 1)); }

  // adjusts value of the k-th element by v (v can be +ve/inc or -ve/dec)
  void adjust(int k, int v) {                    // note: n = ft.size() - 1
    for (; k < (int)ft.size(); k += LSOne(k)) ft[k] += v; }
};
```

Runtime Complexity

Construction	O(M log N)
Query	O(log N)
Point Update	O(log N)

Note:

N is the array size.
M is the number of data points.

Runtime Proof: Query

- A base 10 number N is represented by at most log(N) bits. Assume N is the largest index in our BIT. Let there be an index B <= N.

- A query to index B must also query index B', B'', B''', while $B^i > 0$. N has at most log(N) bits so this can take at most log(N) operations.

Runtime Proof: Update

- A base 10 number N is represented by at most log(N) bits. Assume N is the largest index in our BIT. Let there be an index B <= N.

- An update to index B must also update index B’, B’’, B’’’, until $B^i > N$. N has at most log(N) bits so this can take at most log(N) operations.

Conclusion

- The Fenwick Tree
  - Supports RSQ and update operations
  - O(N) space complexity
  - O(log N) time complexity
  - Very clever use of bitwise indexing

References

- Peter M. Fenwick (1994). "A new data structure for cumulative frequency tables"
  - http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.14.8917
- Competitive Programming 3
  - https://sites.google.com/site/stevenhalim/
- Wikipedia
  - http://en.wikipedia.org/wiki/Fenwick_tree
- Stack Exchange
  - http://cs.stackexchange.com/questions/10538/bit-what-is-the-intuition-behind-a-binary-indexed-tree-and-how-was-it-thought-a

Questions?