

Elias-Fano coding

Compression of a sequence of increasing integers:
no repeated values: the frequency is 0 or 1

The input are n increasing values in $[0, u)$, when $n \ll u$
Elias-Fano can be seen as a compressed representation that
offers efficient random access and NextGEQ queries

Running example: 1,4,7,18,24,26,30,31

NextGEQ(24) -> 24

NextGEQ(25) -> 26

NextGEQ(11) -> 18

Example: $u = 32, \log_2 u = 5, \quad n = 8, \quad \ell = \lceil \log_2(u/n) \rceil = 2$

$\log_2 u$

1 =	000	01
4 =	001	00
7 =	001	11
18 =	100	10
24 =	110	00
26 =	110	10
30 =	111	10
31 =	111	11

$n\ell$

$L = 0100111000101011$

bucket 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7
 $H = 10|11000|100|110|110$

$(\log_2 u) - \ell \quad \ell = 2$

no compression: 40 bits
 EF: 16+16 = 32 bits

Number of buckets: $2^{(\log_2 u) - \ell} = u/2^\ell$

for our choice of ℓ they are $u/2^\ell \leq u/2^{\log_2(u/n)} = u/(u/n) = n$

The total cost of $L + H$ is:

$$n\ell + 2n \approx n \log(u/n) + 2n$$

bits. Why we consider this a compressed format?

The number of subset of n elements out of an universe of size u is $\binom{u}{n}$. Such subsets cannot be all represented with less than $\log_2 \binom{u}{n}$ bits since an encoding of m bits can represent at most 2^m distinct items, so with $m < \log_2 \binom{u}{n}$ we cannot encode $\binom{u}{n}$ distinct subsets

Hence $\log_2 \binom{u}{n}$ is a lower bound to the number of bits needed to represent such subsets. Using Stirling formula we get that

$$\text{for } n \ll u \quad \text{it is} \quad \log_2 \binom{u}{n} \approx n \log(u/n)$$

Hence the Elias-Fano representation exceeds the lower bound by the term $2n$ (2 bits for each input integer)

Elias-Fano decoding

To decode only the i -th entry:

take the ℓ less significant bits from L

to find the most significant bits we need to figure out which bucket contains the i -th entry;
This can be done by reaching the i -th 1 in H and counting how many 0s there are before that 1.
The number of zeros is the bucket id.

We need to efficiently count the number of 0s before the i -th 1. This is done using the **Select** operation (next page)

Given a binary sequence S

$\text{Select1}(i)$ returns the position of the i -th 1 in S (counting from 1):

For example: $S = 0111010010$

$\text{Select1}(1) \rightarrow 2$ $\text{Select1}(2) \rightarrow 3$ $\text{Select1}(4) \rightarrow 6$

we can similarly define the Select0 operation returning the position of the i -th 0:

$\text{Select0}(1) \rightarrow 1$, $\text{Select0}(2) \rightarrow 5$

For a binary sequence S of length t we can build an auxiliary data structure of size $o(t)$ bits supporting Select0 and Select1 operation in constant $O(1)$ time.

Using the Select1 operation over the array H, we can easily count the number of 0s before the i-th 1 computing

$$j = \text{Select1}(i) - i$$

this is precisely the id of the bucket containing the i-th entry

We conclude that the i-th entry of the sequence can be computed as

$$j2^\ell + (L[1 + (i - 1)\ell] \cdots L[i\ell])_2$$

where $(L[1 + (i - 1)\ell] \cdots L[i\ell])_2$ is the value of the ℓ bit binary number consisting of entries from $1 + (i - 1)\ell$ to $i\ell$ of L

Additional operation we support on Elias Fano encoding:

NextGEQ(i): the smallest integer x in the input, with $x \geq i$

Example: To compute NextGEQ(25) write 25 in binary: $(11001)_2$ so 25 would be stored in bucket $(110)_2 = 6$.

NextGEQ(25) is in bucket 6 or in a subsequent bucket. To check such values observe that $\text{Select}_0(6)+1 = 11$ is the position of the first entry in H of bucket 6.

1 =	000	01
4 =	001	00
7 =	001	11
18 =	100	10
24 =	110	00
26 =	110	10
30 =	111	10
31 =	111	11

$L = 0100111000101011$

bucket	0	1	2	3	4	5	6	7
$H =$	10	11	00	01	00	11	10	11

Another example: NextGEQ(11)

$$11 = (01011)_2$$

so we have to look at bucket $010 = 2$, or later.

The first bit in H of bucket 2 is the one in position

$$\text{Select0}(2)+1 = 6$$

see textbook and python code for additional details

1 =	000	01
4 =	001	00
7 =	001	11
18 =	100	10
24 =	110	00
26 =	110	10
30 =	111	10
31 =	111	11

$$L = 0100111000101011$$

bucket	0	1	2	3	4	5	6	7
H	10	110	00	100	110	110		