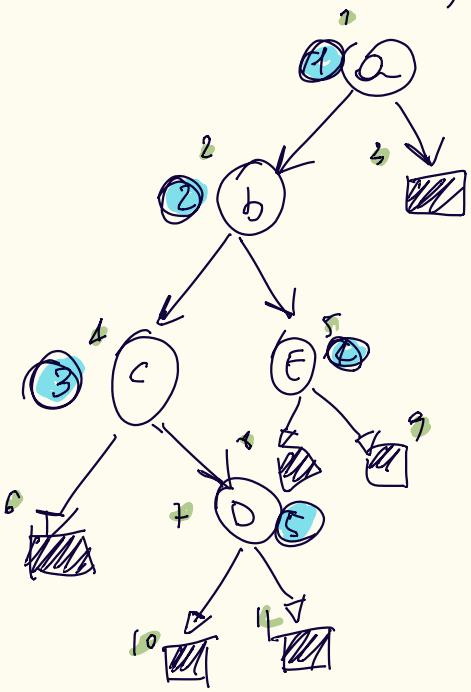
Succint encoding of TREES

trees provided in the following: $(a,b)(b,c)(b,\varepsilon)(c,d)$ (ε = e, R)

(edges, a b c d labels of nodes)



- complete binary tree with dummy leaves

=> LOUDS representation

- start from the root
- print: 1 for real nodes, 0 for dummies

	1	2	3	4	5	6	7	8	9	10	11
	1	1	0	1	1	0	1	0	0	0	0
# real nodes	1	2		3	4		5				

↑ c

- given this encoding, jump from (a) to (b):

(find left child):

$2 \cdot X$

STARTING NODE = ON the left level

- read look if is 1 or 0 → no child

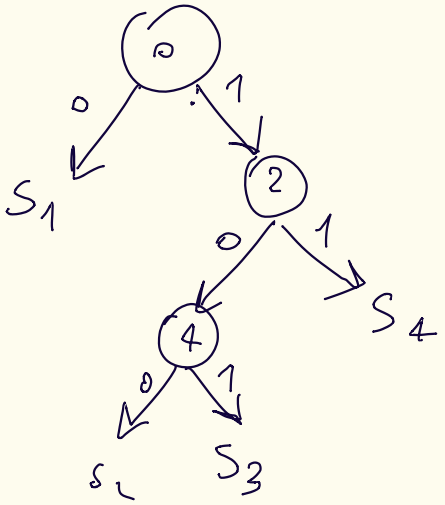
if 1 => do a rank

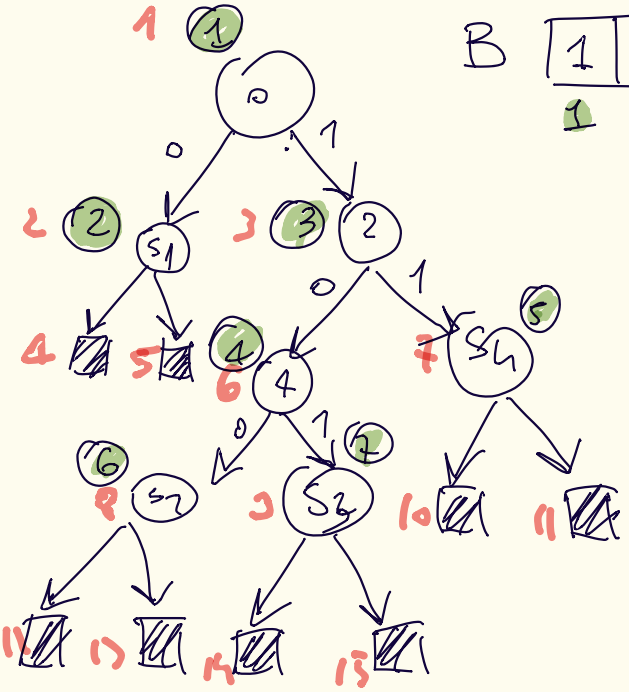
es. for (c) = 4 $\quad 2 \cdot 4 = 8$ is zero => no child

es. for (a) = 1 $\quad 2 \cdot 1 = 2$ 2 has 1 => has a child

=> do a rank

How to represent the tree in a succint way





B

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	1	0	0	1	1	1	1	0	0	0	0	0	0
1	2	3			4	5	6	7						

infos about nodes and their labels

L

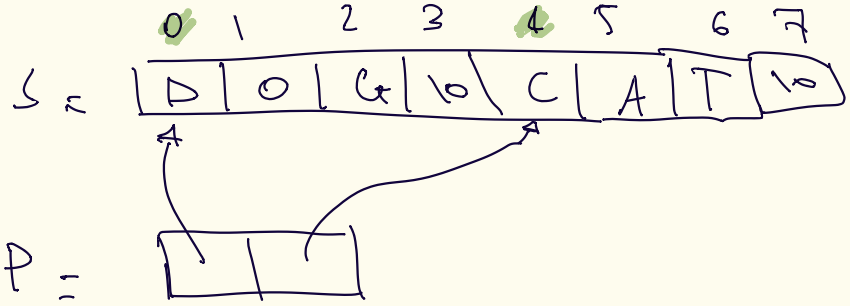
1	2	3	4	5	6	7
0	S_2	2	4	S_4	S_2	S_3

$Rank_1(2x)$

TREE ENCODING AND TRAVERSAL

SVCL INT
not compressed

Problem: store strings in an array and keep pointer to the strings



array of pointers: - full-size pointers: 4/8 bytes => P $O(n \cdot w)$ bytes (w: 32/64 bits)

or

- keep the offset => P' = [0|4] offset to the beginning of X

=> WE WANT STORING OFFSET not POINTERS → Elias-gamma ↓

sort increasing integers

$O(2n + n\lceil \log \frac{m}{n} \rceil + o(n))$

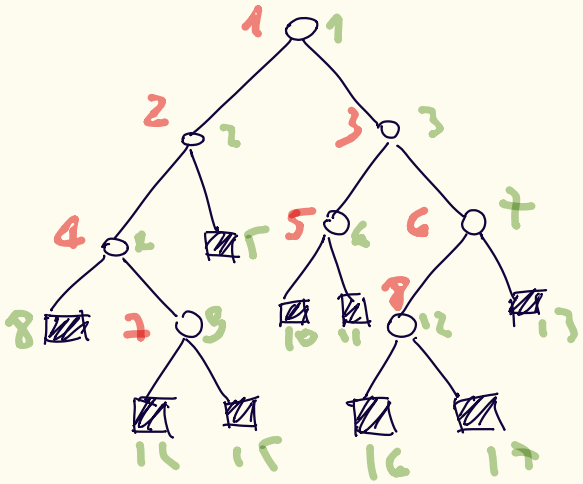
where $m = |S|$ universe

- Give me the 2nd STRING: access(2)

TREES with no pointers

- Binary tree

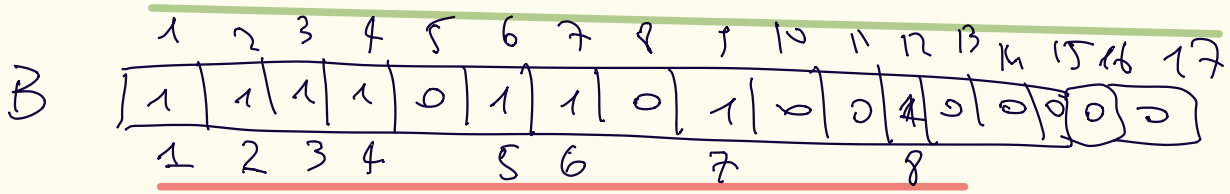
- n = 8 (nodes)



1) Tree completion

2) - numerate real nodes
   - numerate all

3) STRATIFICATION of node label



- On B => compress + index B using rank & select DS

space $O(|B| + o(|B|) + o(|B|)) \Rightarrow O(2n+1 + o(n))$

($o(|B|)$: $rank_1$, $o(|B|)$: $select_1$)

$|B| = 17 = 2n+1 = 2 \cdot 8 + 1 = 17$

$\downarrow$

# of original nodes

- TRAVERSE: from a node (x) => get a **left_child**: $2x$ (all nodes) => $Rank_1(2x)$ if $B[2x] == 1$

if $B[2x] == 0$ || $B[2x+1] == 0$ => dummy child no exist

* dummys don't exist

I've to do a step back => transform dummy into real value

$\downarrow$

$rank_1(x)$ in B array

from a node (x) => get a **right_child**: $2x+1$ (all nodes) $Rank_1(2x+1)$ if $B[2x+1] == 1$

from a node (x) => get **the parent**:

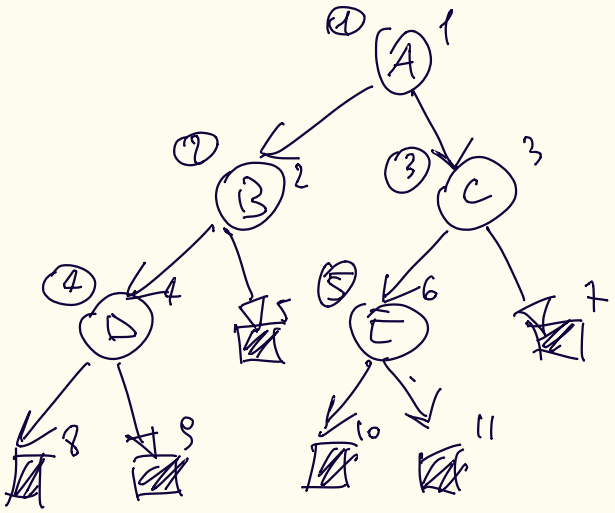
$$\left\lfloor \frac{Select_1(x)}{2} \right\rfloor$$

iff $x \neq 1$ $\downarrow$ we're not in the root

miss che è

$$root \left\lfloor \frac{select_1(x)}{2} \right\rfloor$$

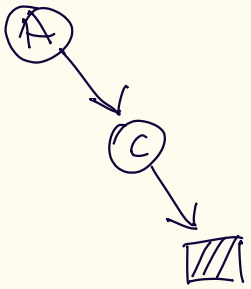
- Given a Tree: $\{(a, b)_L\ (a, c)_R\ (B, D)_L,\ (C, E)_L\}$
build the tree encoded in succint way and show a tree traversal from the root to the right child, right child



B =

1	2	3	4	5	6	7	8	9	10	11
1	1	1	1	0	1	0	0	0	0	0

Tree traversal: ROOT → RIGHT CHILD → RIGHT CHILD



(1)

$Rank_1(2x+1) =$

$= Rank_1(3)$

iff: $B[3] == 1$ (OK)

(3)

$Rank_1(2x+1) =$

$Rank_1(7)$

$B[7] == 0$ → MISBA

=> I'm in a Dummy node

=> NULL

- Which are the level of the node traversed by the procedure? (we want the label of the nodes)

=> need a Dictionary that associates to every number (1, 2, 3, 4, 5) the corresponding letter

L (labels) =

1	2	3	4	5
A	B	C	D	E

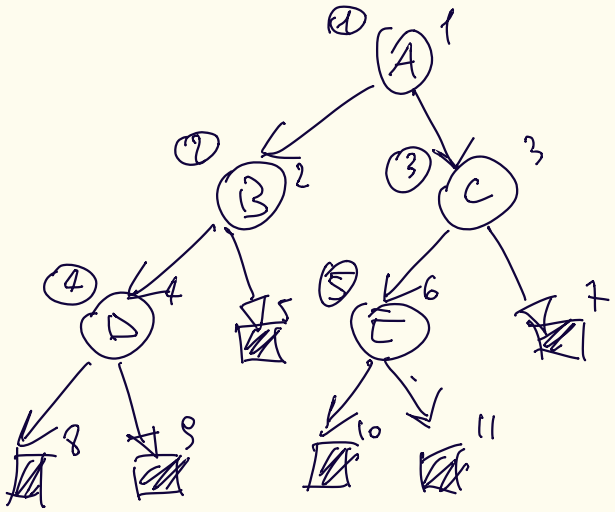
root $L[1] = A$

right child $L[3] = C$

2nd right child $L[7] = NULL$ ✓

- Given a sequence of labels, want to see if ∃ a path labeled from ① to ⑪ in the tree and traverse the path

WRITE THE PSEUDO CODE



B =

1	2	3	4	5	6	7	8	9	10	11
1	1	1	1	0	1	0	0	0	0	0

L = 1 2 3 4 5

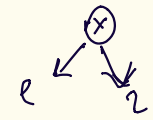
- The path is given by an array Π: [A|C] => yes
  Π [A|B|C] => no

- Match (Π, ...) -> parameters of the tree)

Π =

1	2	3
a		

```
x = 1;                                   // start from da root
if (L[1] ≠ Π[1]) return false;           // check if the label of the root is the same of Π[1]
For (i = 1 to |Π| - 1) do                // -> root already done
                                         // check right/left child Exists
    ℓ = rank_1(2x);
    r = rank_1(2x+1);
if (B[2x] == 1 && (L[ℓ] == Π[i+1]))      // left child ∃, and label pos. ℓ is equal to next label of path I'm checking
    then x = ℓ;                          // ℓ -> left child
else
    if (B[2x+1] == 1 && (L[r] = Π[i+1])) // check right child
    Then x = r;
ELSE RETURN FALSE                        // exist cuz if & else are fake
```



}

(OUT OF THE FOR)

RETURN TRUE

- Given the binary tree : (a,b)(a,c)(c,d)(c,e)(e,f)(f,g)

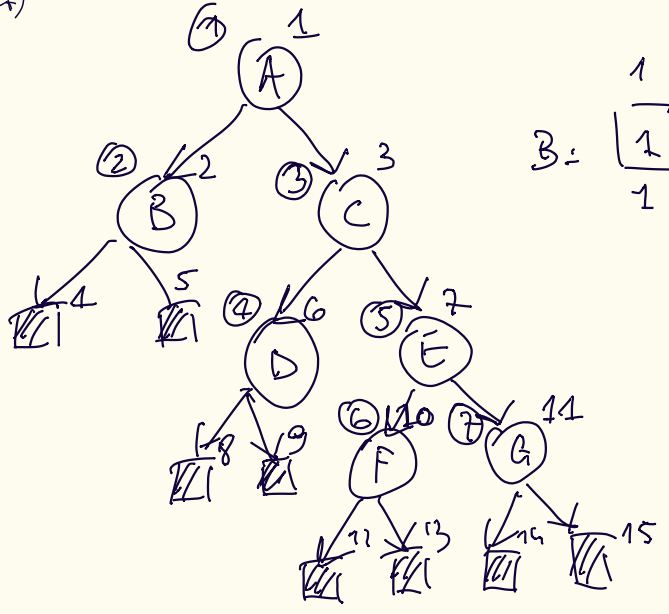
1) encode the tree using LOUDS

2) Simulate the path traversal ACD with rank, select DS

3) Write pseudocode that takes as input a binary tree T with nodes labeleded with letters (possibly repeated) and represented via LOUDS plus an array L of those letters, and takes in input also a letter X and a positive integer d, prints true if ∃ a node in T labelled x and lvs a distance from the root (distance = # of edges). You have rank, select

(a,b)(a,c)(c,d)(c,e)(e,f)(f,g)

1)

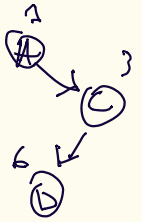


B: 1 1 1 0 0 1 1 0 0 1 1 0 0 0 0

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

1 2 3 4 5 6 7

2) Path: ACD



1) START from the root and I go to a rightchild, which is real node if B[2x+1] = = 1

x = (A) = 1

2x+1 = 3

B[3] = = 1 ok

=> rank$_1$[3] = = 3

from node x=3 I go to a left child if B[2x] = = 1 B[6] = = 1 ok

rank$_1$[6] = = 4 ok

3) Procedure α (T, L, X, d)

n = # of nodes of T

for i = 1 to n do

if (L[i] == X) then

y = i; distance = 0;

while (y > 1)

y = L[select$_1$($\frac{y}{2}$)]

distance ++;

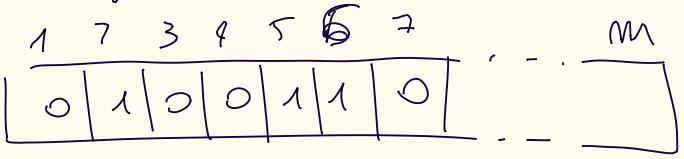
if (distance == d)

return true

RETURN FALSE

COMPRESSED DS

- given a Binary array



- $m = |B|$
- $n = \#1s$

• $M = |U|$ ← universe size

- WANT STORE positions of 1

⇒ STANDARD ENCODING: space $O(\#1s \cdot$ bits to represent position of 1s$) =$
$= O(n \cdot \log_2 m)$ bits

→ Pointer-based programming

⇒ want pointer-less programming: idea: 2 operations

1) $\text{rank}_b(i)$ (i → position; b ↑ bit $\{0,1\}$) = # of b in the prefix $B[1,i]$

⇒ $\text{rank}_1(4) = 1$
$\text{rank}_1(5) = 2$

$\text{Rank}_0(i) = i - \text{RANK}_1(i)$

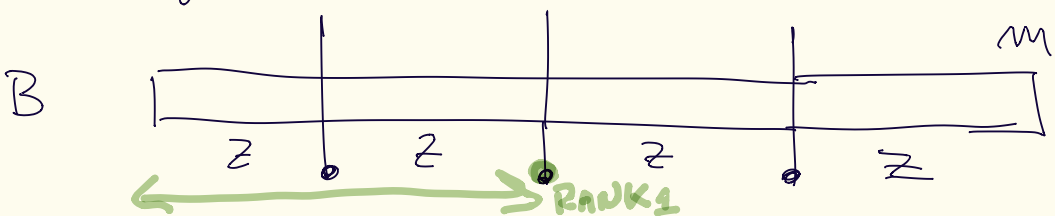
2) $\text{select}_b(i)$ = position of i-th b in B

b ↑ bit $\{0,1\}$ ⇒ $\text{select}_1(3) = 1$ ⇒ 3rd 1

• THEO: ∃ a DS that takes $o(m)$ (→ something that asymptotically is $< m$) bits in addition to (B) and supports rank_1 in $O(1)$ time

(B) → read only (not touched)

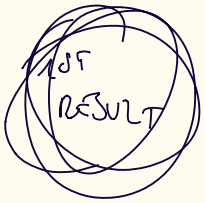
1) partition array B into blocks of size Z



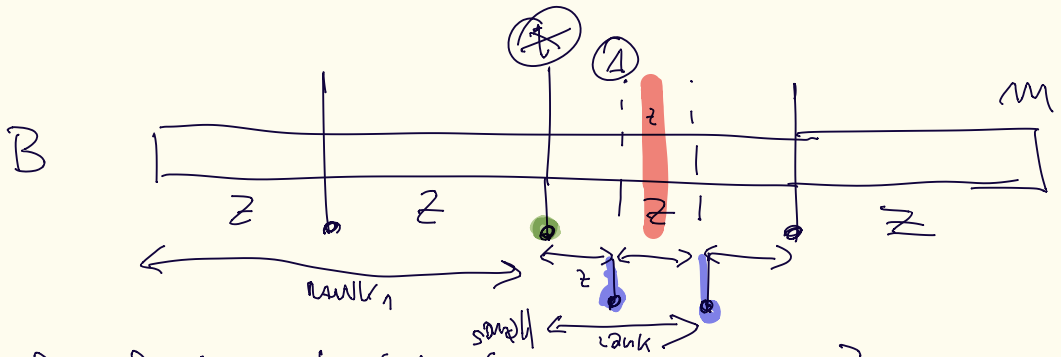
big blocks $z = O\left(\frac{m}{z}\right)$

2) $\forall$ block z, I store $rank_1$ of all previous blocks (ABSOLUTE: from the start)

- Space of Big blocks = # big blocks · counter ($rank_1$) $\Rightarrow O\left(\frac{m}{z} \cdot \log_2 m\right)$ bits



- ANOTHER PARTITION



- Every Big Block is divided in $\frac{z}{t}$ Blocks $\Rightarrow \frac{z}{t}$

- $\forall$ small block I store: RELATIVE $rank_1$

- space for small block t: $O\left(\frac{m}{z} \cdot \log_2 z\right)$ bits

$\Rightarrow$ want $rank_1$ here: $rank_1$ of z (*) $O(1)$
+ $rank_1$ of z (A) $O(1)$
and scan the small block $O(z)$

$\Rightarrow$ TOTAL SPACE: $O\left(\overbrace{\frac{m}{z}\log_2 m}^{\text{Big blocks } z} + \overbrace{\frac{m}{z}\log_2 z}^{\text{small blocks } z}\right)$ bits

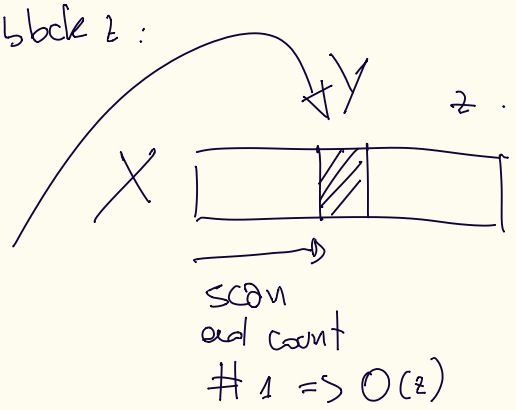
Time Complexity: $O(z)$ $\Rightarrow$ but we want constant time

(SCAN) small block

3RD-step : 4-Russian trick => we're in RAM model with word size $O(\log n)$ bits
(=> every operation on $\log n$ bits is constant time $(-; +; \cdot; \div)$)

- given small block z:

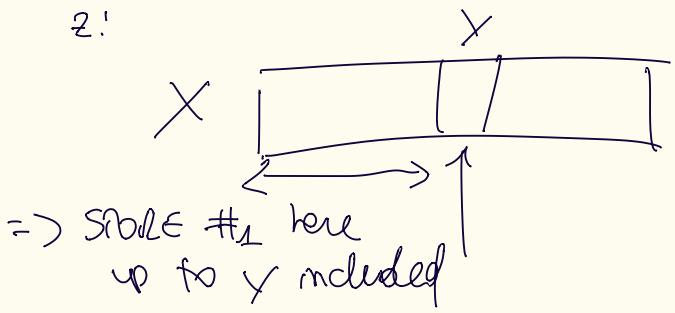
- Assume: I'm querying the position



=> to do in constant time, I tabulate all possible answer
=> brute force approach

=> $\forall$ small block z configuration (call it x) and $\forall$ position $y \in [1, z]$

=> we store $\#_1$ in $x[1, y]$



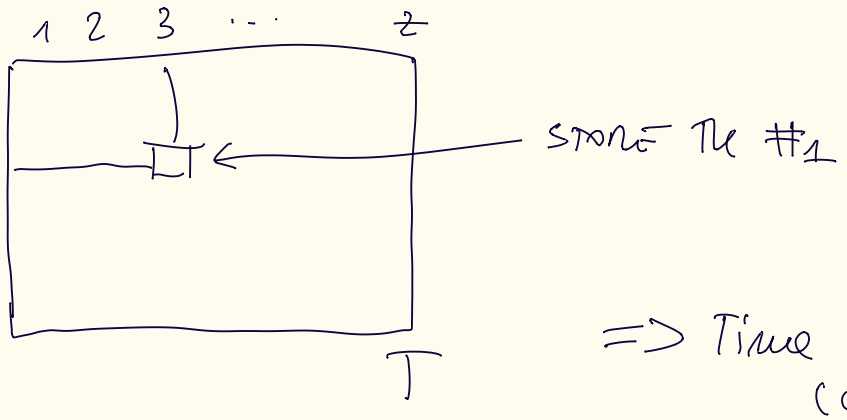
- Create a table:

all block X

000 ... 00

...

111 ... 11



=> Time is $O(1)$ (constant)

=> Space of the table:

I've 2^z rows $\cdot$ z columns and every entry requires $\log_2 z$ bits

Total complexity of the space:

$$O\left(\frac{n}{z}\log n + \frac{n}{z}\log z + \underbrace{2^z \cdot z \cdot \log_2 z}_{\text{TABLE}}\right) \text{ bits}$$

query $\Rightarrow$ $rank_1(i) =$

- $\alpha(i)$ → counter stored at z including i
- $\alpha(i)$ → counter stored at z including i
- $o(n)$ → let X be the small block including i
- α_1 → let $y = i \bmod z$, get $T[X, Y]$ $\Rightarrow$ constant

this is the problem:

- set $z = (\log_2 n)^2$
- set $z = \frac{1}{2}\log_2 n$ $\Rightarrow$ half of memory word

$$O\left(\frac{n}{z}\log n + \frac{n}{z}\log z + 2^z \cdot z \cdot \log_2 z\right) =$$

$$= O\left(\frac{n}{\frac{1}{2}\log_2 n}\log n + \frac{n}{\frac{1}{2}\log n} \cdot \log\left(\log(n)\right) + 2^{\frac{1}{2}\log n} \cdot \frac{1}{2}\log n \cdot \log\left(\frac{1}{2}\log n\right)\right) =$$

$$= \frac{n}{\log n} + \frac{n}{\log n}\log\log n + \underbrace{2^{\frac{1}{2}\log n}}_{2^{\log(n)^{\frac{1}{2}}} = 2^{\log\sqrt{n}} = \sqrt{n}} \cdot \log n \cdot \log\log n =$$

$$= \frac{n}{\log n} + \frac{n}{\log n}\log\log n + \underbrace{\sqrt{n} \cdot \log n \cdot \log\log n}_{o(n)}$$

$\Rightarrow o(m) \leq \frac{m}{10}$ = 10% of universe in real scenario

Have to compare with SCAN solution

#A $\xrightarrow{O(n \log m)}$

=> still depends on n

SELECT

THEO : $\exists$ DS that uses $o(m)$ bits in addition to B that support $select_1$ and $select_0$ in $O(1)$ time

ND

=> This and other theorem do not depend on $(n, keys, \Delta s)$ but on Universe size

=> HAVE TO COMBINE THOSE 2 THEOREMS WITH ELIAS-FANO COMPARISON

- take Elias-Fano: compare it with $select_1$ and $select_0$

=> I get a DS that occupies:

space $2m + n\left\lceil \log \frac{m}{n} \right\rceil + o(m)$ BITS (# keys)

AND Can support

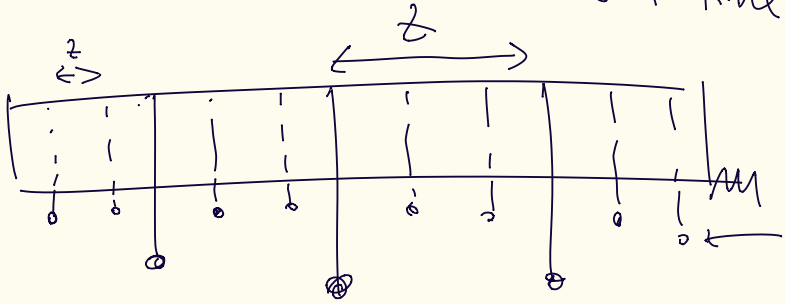
$Rank_1$ in $O(\log \frac{m}{n})$ time

$select_1$ in $O(1)$ time)

THEOREM: ∃ DS that takes $o(m)$ additional to B that support Rank in $O(1)$ time

constant time

Idea



absolute Rank 1

Big Blocks $z \Rightarrow O\left(\frac{m}{z} \cdot \log_2 m\right)$

Small Block $z \Rightarrow O\left(\frac{m}{z} \log_2 z\right)$

=> 4-RUSSIAN TRICK: store a table with ∀ possible config of small blocks z stores the values of 1s of positions

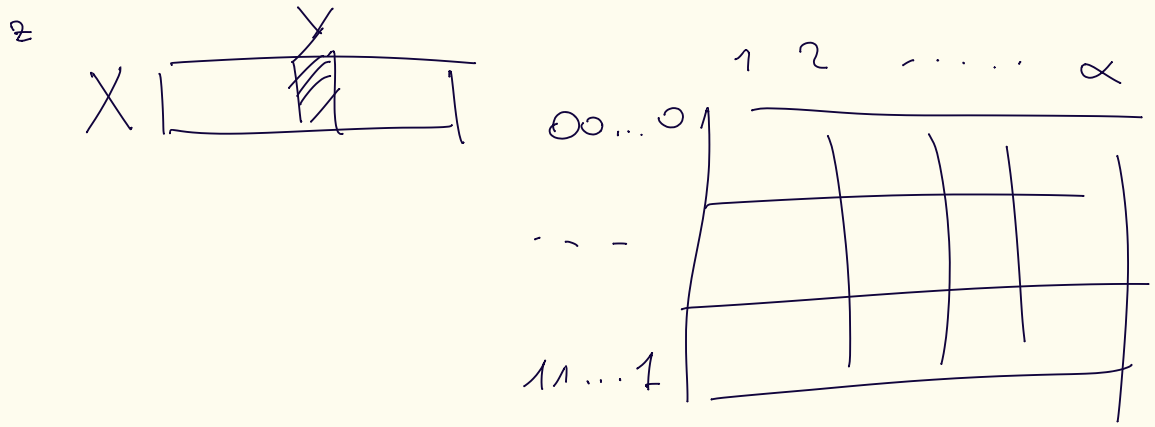


Table requires

ROWS · COLUMN · REPRESENTATION

$2^z \cdot z \cdot \log_2 z$

=> Complexity becomes

$$O\left(\frac{m}{z} \log m + \frac{m}{z} \log z + 2^z \cdot z \cdot \lg z\right)$$

- set $z = (\log_2 m)^2$

- set $z = \left(\frac{1}{2}\log m\right)$ half of a word

SOBSTITUTE THAT SHIT

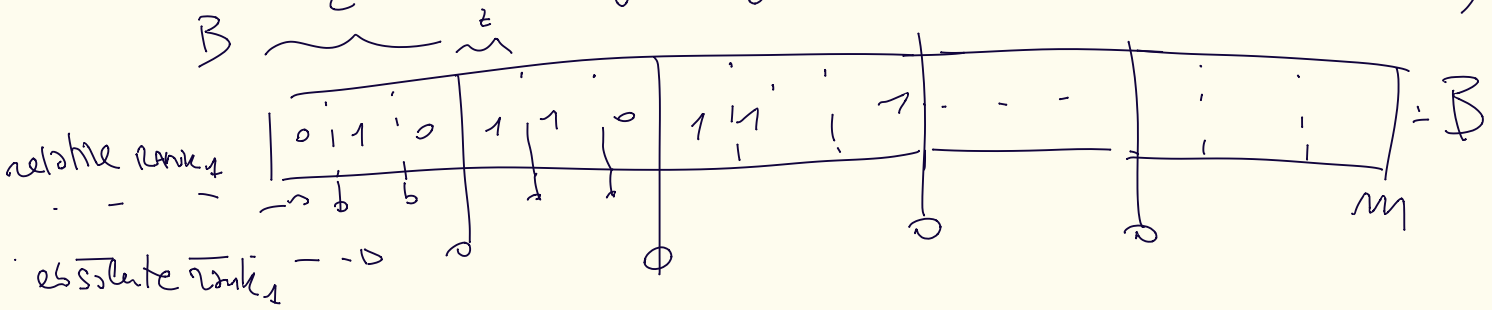
$\Rightarrow$

$$\frac{m}{(\log_2 m)^{\chi}} \log m + \frac{m}{\frac{1}{2}\log m} \log\left(\log m\right)^{\chi} + 2 \cdot \frac{1}{2}\log m \cdot \log\left(\frac{1}{2}\log m\right)$$

$\frac{1}{2}\log m \le \sqrt{m}$

$O(m)$ $\quad$ $O(m)$ $\quad$ ✓ $\quad$ $O(m)$

THEOREM : ∃ a DS that takes $\mathcal{O}(m)$ bits in addition to a Binary vec B that supports rank in $O(1)$



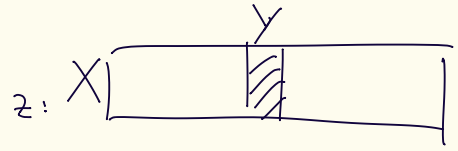
z : Big Blocks : $\mathcal{O}\left(\frac{m}{z} \log m\right)$ bits

z : Small Blocks : $\mathcal{O}\left(\frac{m}{z} \log z\right)$ bits

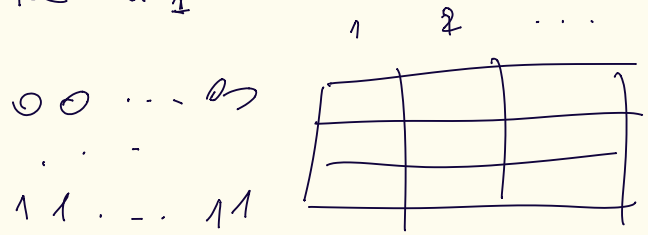
$rank_z(i)$ returns # 1 up to i (i indexed)

TABLE: (4 Russians Trick):

=>



- table stores for every possible configuration of small block z the # 1



rows · colums · representation

$2^z \cdot z \cdot \log_2 z$

$$\Rightarrow \mathcal{O}\left(\frac{m}{z} \cdot \log_2 m + \frac{m}{z} \log_2 z + 2^z \cdot z \cdot \log_2 z\right)$$

=> set $z = (\log_2 m)^2$

$$z = \frac{1}{2} \log m$$

=> substitute

$\log(i) = \log_2(i)$

$$O\Big(\underbrace{\frac{n}{(\log n)^2} \cdot \log n}_{\sigma(n)} + \underbrace{\frac{n}{\frac{1}{2}\lg n} \log(\lg n)^2}_{\sigma(n)} + \underbrace{2^{\frac{1}{2}\lg n} \cdot \frac{1}{2}\lg n \cdot \lg_2\left(\frac{1}{2}\lg n\right)}_{\sigma(n)} \Big)$$

$$= \sigma(n) \quad (\checkmark)$$