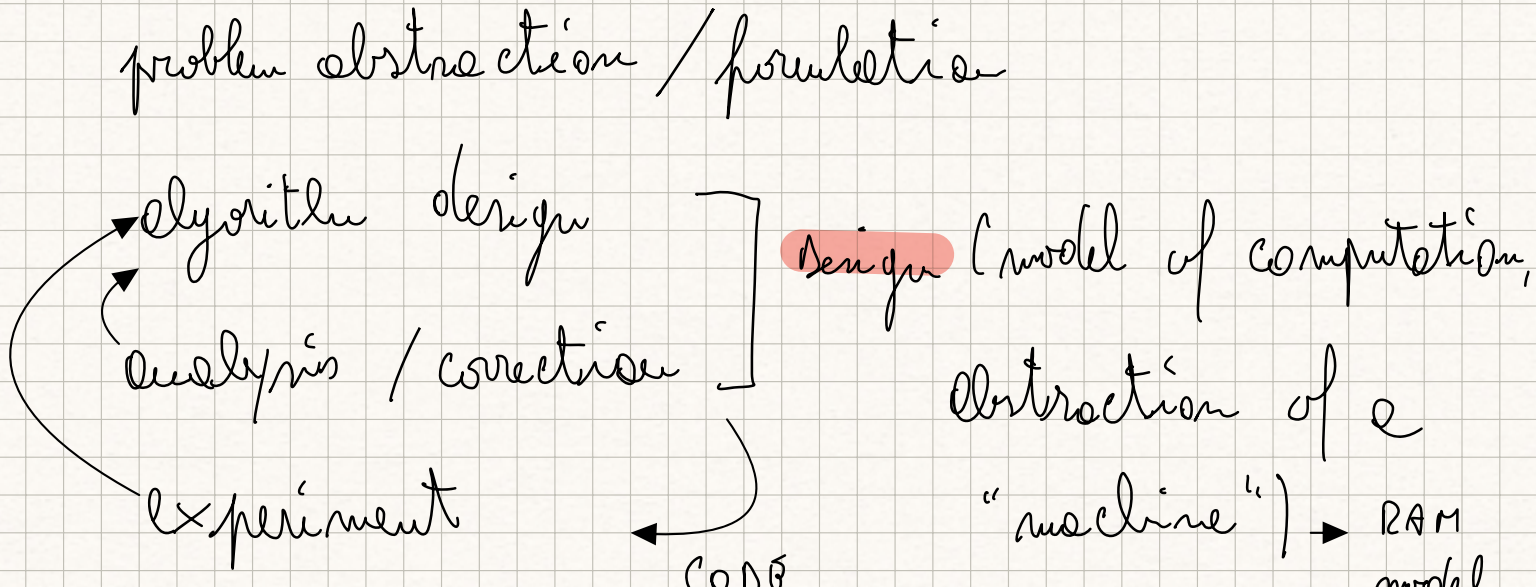
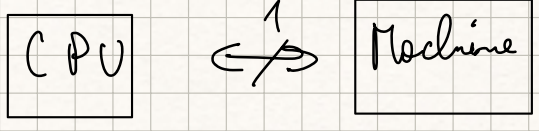
Algorithm: Comunication of the ACM

Def. of algorithm (by Knuth)

- Finite seq. of steps
  - w prop:
    - DEFINITE = unambiguous → programming lang.
    - EFFECTIVE = f: input → output ∀ I
    - FINITE → O(1) TIME
    - INPUT and OUTPUT



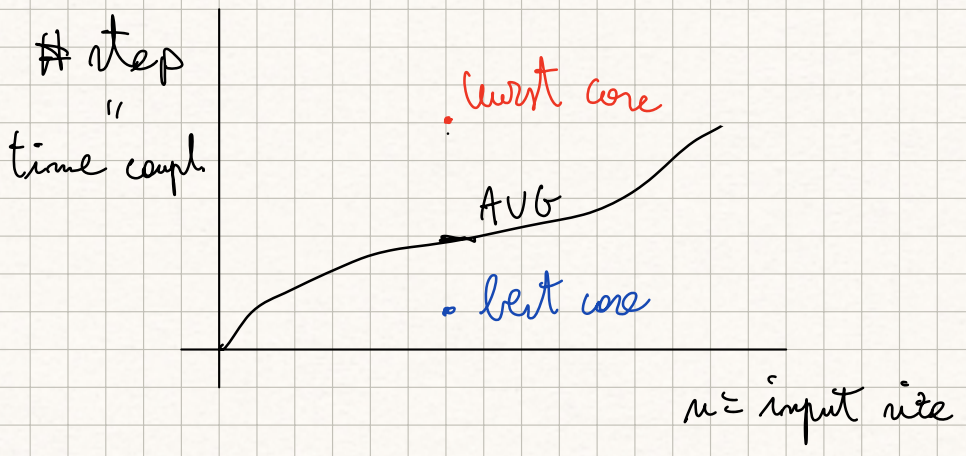


• time complexity of an alg. $A \simeq$ # steps (comparison)

↓

y(Input)

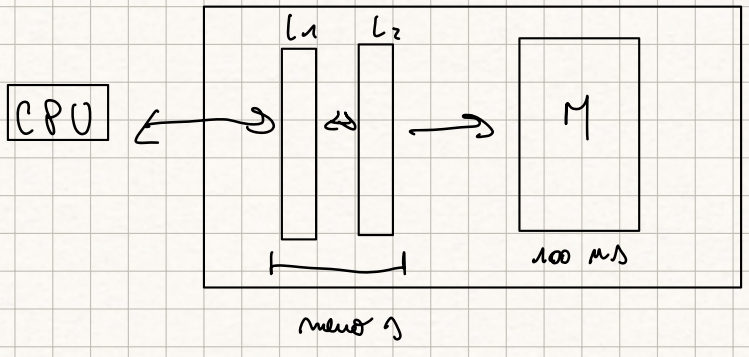
↳ considero y di input che mi danno il worst case



• to compare alg A_1 and A_2 → # steps

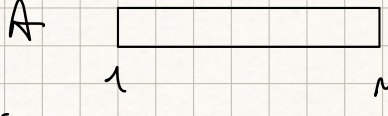
for BIG DATA $n \to \infty$

• memory → hierarchy of memory levels



→ extremely complex. We need approximation.
es. external memory model

es:



sum(A) : temp = 0

for i = 1 to n

temp $\leftarrow$ A[i] / O(n)

RAM. old alg.

A S, b

↓ Jump ↘ block size within the Page B

$A_j = A[j \cdot b + 1, (j+1) \cdot b]$ → logical block

$A_0 = A[1, b]$

$A_1 = A[b+1, 2b]$

if $S = 1$ ⇒ scanning

· seq of block I occes

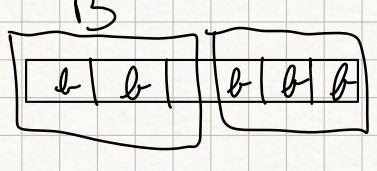
$j = S \cdot i \bmod \frac{n}{b}$ $\quad i = 0, 1, \dots, n/b - 1$

$S >> 1$ → different pattern of memory access

↳ co-prime with $\frac{n}{b}$ (per avere ricorso che ha accesso a tutti gli item)

↓

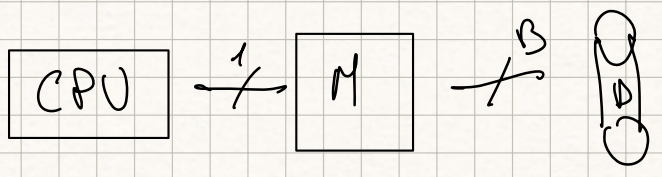
M.C.D. = 1

numb of mex pass → B/b A: 

$O(I/O) = \min\left\{S, \frac{B}{b}\right\} \cdot n/B$

$O(\text{time}) = n$

Two - level memory model

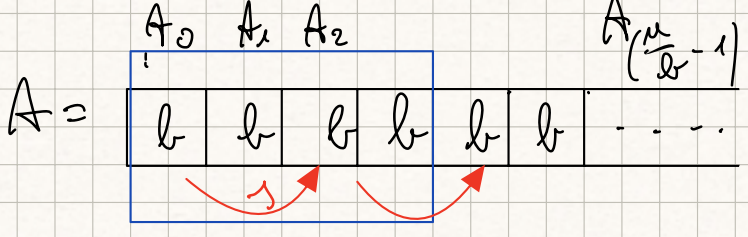


internal memory / HDD SSD

cache / internal memory

disk / network

two level memory depends on how much data i have



B = page size

design algorithm → choose b, s

analysis system → fixed M, B

A_j where $j = i \cdot s \bmod \frac{n}{b}$

$i = 0, 1, 2 \dots n/b - 1$

Time complexity $= O(n)$

I/O complexity $= O\left[\min\left\{\frac{B}{b}, s\right\} \cdot \frac{n}{B}\right] =$

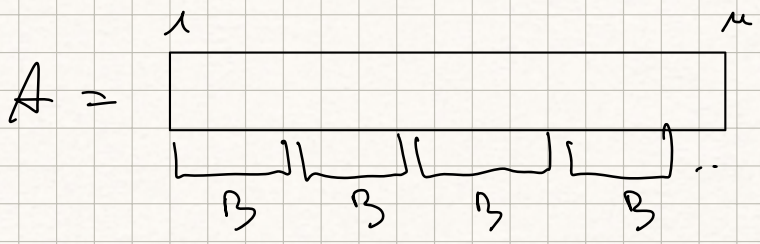
$s = 1 \rightarrow \frac{n}{B}$ scanning

$s > 1 \rightarrow s \cdot \frac{n}{B}$

$s < \frac{B}{b}$

Searching in a sorted array

$x \in A$?



binary search: comparison based algorithm

Time $= O(\log_2 n)$

I/O $= \log_2 n - \log_2 B = \log_2 \frac{n}{B}$

↓ cache regime — ↓ non cache regime

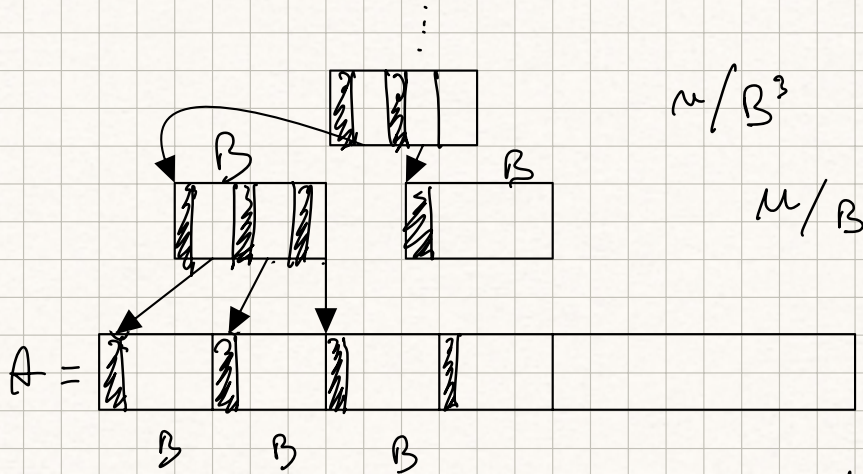
$B = 32$ Kb ← seek time 5ms, fetch time

8 byte ints

items × page $= \frac{2^{15}}{2^3} = 2^{12} \approx 4000$ items $\log_2 \frac{2^{30}}{2^{12}} = 18$ I/O

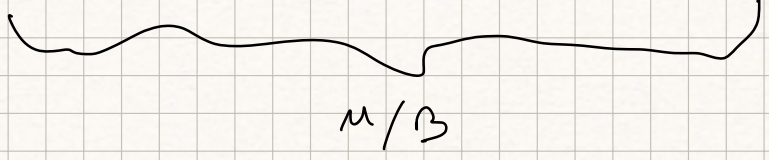
$\geq B$

B-Tree:



n/B^3

n/B^2



the root will be $\frac{n}{B^l} \leq 1 \rightarrow \log_B n \leq l$

I/O complexity $= \log_B n = \frac{\log_2 n}{\log_2 B}$

$\frac{\log_2 2^{30}}{\log_2 2^{11}} = \frac{30}{11} \simeq 3$ I/O much better than before

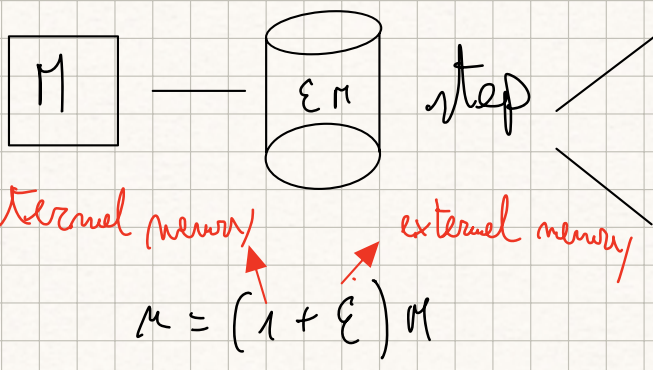
$\log_2 \frac{n}{B} \iff \log_B n$

(18) (3)

B-tree is static, meaning that i can move inside it with arithmetic (no pointer)

Spatial locality : i use items inside the same block

Temporal locality: i use during the execution items of the same blocks



step
- computation $1-\alpha = 70\%$ — cost = 1
- $\alpha = 30\%$ memory access
  - internal — cost = 1
  - external — cost = c

prob. of taking random an item in the external memory:

$$P(\varepsilon) = \frac{\varepsilon \cdot M}{n = (1+\varepsilon)M} = \frac{\varepsilon}{1+\varepsilon}$$

$\varepsilon \cdot M$: data on disk (B

$n = (1+\varepsilon)M$: all the data

every cost step: $(1-\alpha)\cdot 1 + \alpha\left[\left(1-P(\varepsilon)\right)\cdot 1 + P(\varepsilon)\cdot c\right]$

- $(1-\alpha)$: comp. cost, < 1
- 1: internal mem. access
- $P(\varepsilon)$: external memory access, < 1

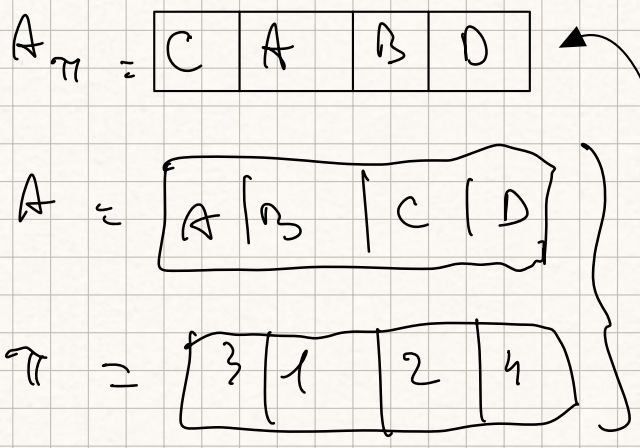
$$< \alpha + \alpha \cdot P(\varepsilon)\cdot c \simeq 0.7 + 0.3 \cdot \frac{\varepsilon}{1+\varepsilon} \cdot 10^6 \simeq \simeq 10^5 \cdot \underbrace{\frac{\varepsilon}{1+\varepsilon}}_{P(\varepsilon)} + \text{cost}$$

$P(\varepsilon) = \frac{1}{1000}$ accessing to the external memory could cost a lot but with low probability $P(\varepsilon)$

Sorting

	RAM
permuting	n
sorting	$n \log n$

SORTING



for $i = 1$ to n

$A_\pi[i] = A[\pi[i]]$

- compute the sorted permutation π
- implement π

$\Big\}$ $n \cdot \log n$

$O(\cdot)$	RAM	2-level (external memory)
permuting	n	$\min \left\{ n, \frac{n}{B} \cdot \text{sort} \right.$
sorting	$n \log n$	sort

I/Os: $\begin{cases} 3 \text{ scan} \\ 2 \text{ sort} \end{cases}$ $O\left(\frac{n}{B}\right)$

Reduce the permuting problem to a sorting problem

Memory → File of tuples

sequence

error

$S = \{ \langle A, 1 \rangle, \langle B, 2 \rangle, \langle C, 3 \rangle, \langle D, 4 \rangle \}$ cost: $O\left(\frac{n}{B}\right)$ I/Os

item position

$\pi = \langle 3,1 \rangle \langle 1,2 \rangle \langle 2,3 \rangle \langle 4,4 \rangle$ cost: $O\left(\frac{n}{B}\right)$ I/Os

source destination

2) Sort π by first element: $\langle 1,2 \rangle \langle 2,3 \rangle \langle 3,1 \rangle \langle 4,4 \rangle$

3) Scan π and S: $\langle A,2 \rangle \langle B,3 \rangle \langle C,1 \rangle \langle D,4 \rangle$

4) Sort π by second component $\langle C,1 \rangle \langle A,2 \rangle \langle B,3 \rangle \langle D,4 \rangle$

$$Sort(n, M, B) = O\left(\frac{n}{B} \cdot \log_{\frac{M}{B}} \frac{n}{M}\right) \text{ I/Os}$$

elements, memory, page size

$\frac{n}{B}$: scan

$M = 16\,GB = 2^{34}$ $\quad \frac{M}{B} = 2^{19}$

$B = 32\,KB = 2^{15}$

$$\log_{\frac{M}{B}} \frac{n}{M} = \frac{\log_2 n/M}{\log_2 M/B} = L$$

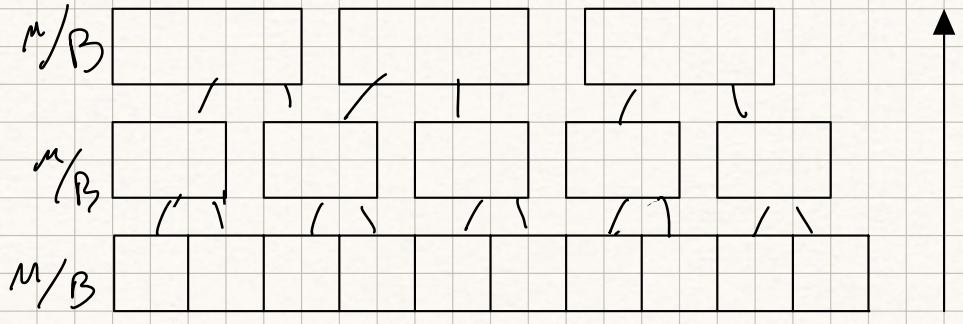
$n = 2^{60}$

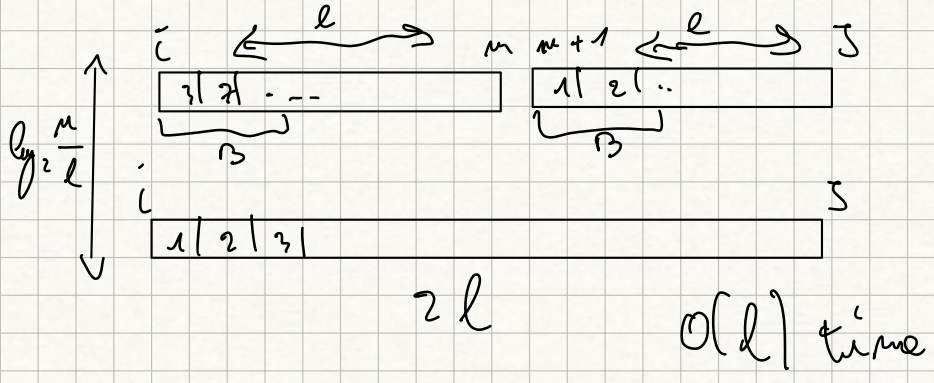
$$\#\text{scans} = \frac{\log_2 \frac{2^{60}}{2^{34}}}{\log_2 2^{19}} = \frac{26}{19} \approx 2$$

When permute is better than sort!

$\frac{n}{B} \cdot L \leq n \longrightarrow B \geq L$ L is 2 so permuting and sorting are equal

Binary Mergesort:





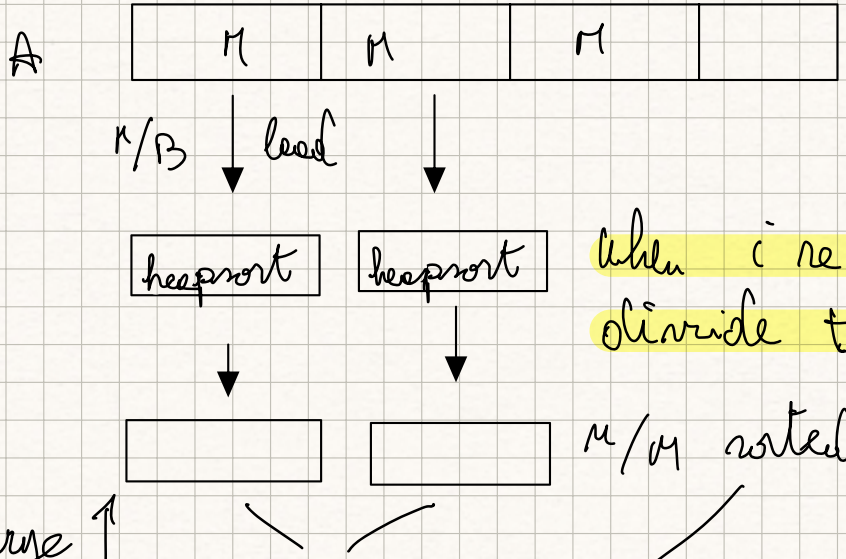
- $O\left(\frac{l}{B}\right)$ I/O to read
- $O\left(\frac{l}{B}\right)$ I/O to write

Binary MS $= O\left(\frac{N}{B} \log_2 N\right)$ I/O

Considering Memory $= O\left(\frac{N}{B} + \frac{N}{B}\left(\log_2 N - \log_2 B\right)\right) = O\left(\cancel{\frac{N}{B}} + \frac{N}{B}\left(\log_2 \frac{N}{B}\right)\right)$

$l \leq B$ $\quad$ $l > B$

dim. of the partition l less or eq. to block size

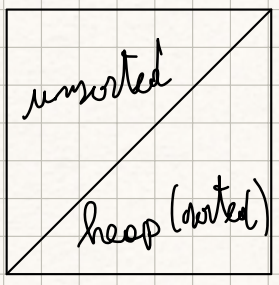


When i reach the size of M i stop dividide the blocks, the i can use heaps.

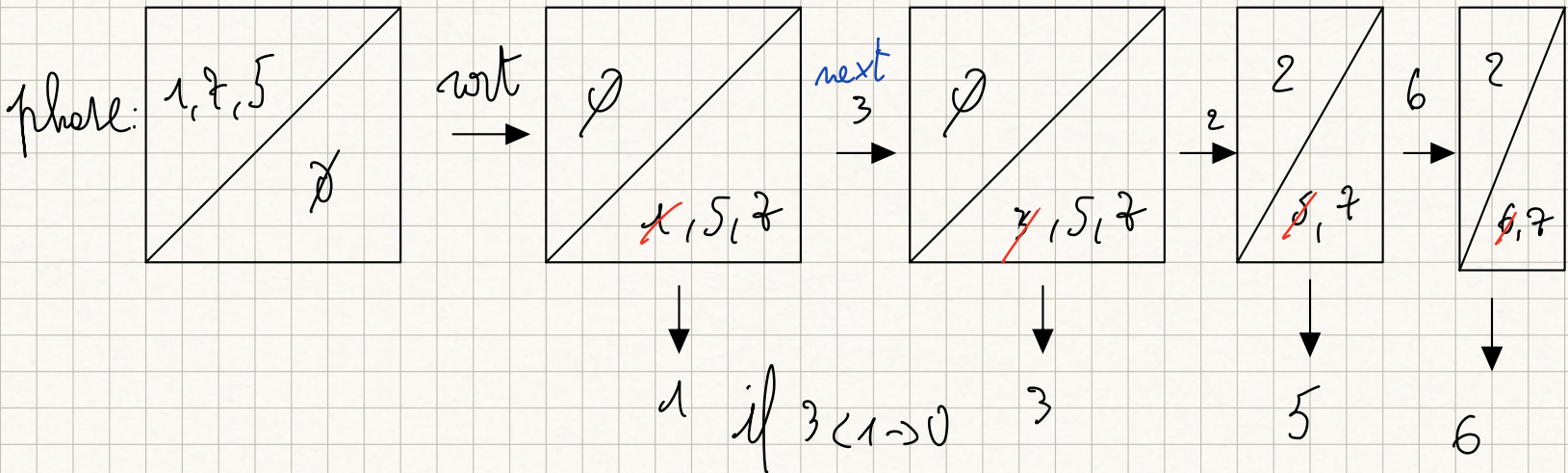
$\log_2 \frac{n}{2M}$

Snow Plow:

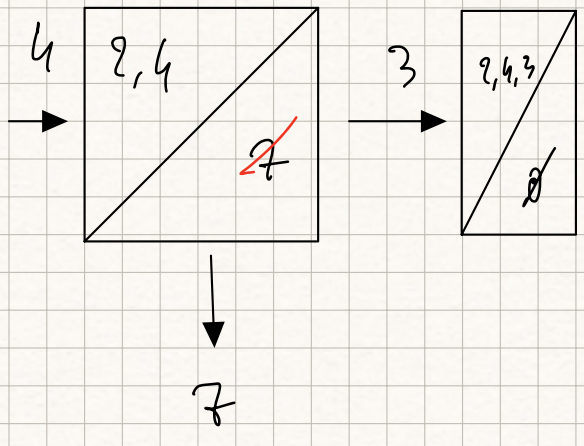
$M = 3$



A = | 1 7 5 | 3 2 6 | 4 3 ... |



1 3 5 6 7



if $3 < 1 \rightarrow 0$

if $3 \geq 1 \rightarrow 4$

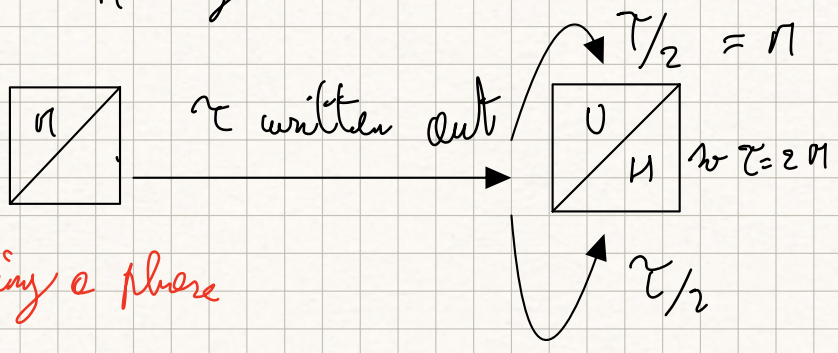
the expected length of a run created by Snow Plow is $2M$.

- load M items in memory 

- load τ items until the heap is empty and (unsorted) U if full again

P(item goes to H) = 1/2

- fetched $\tau + M$ items $\tau + M$ → [M] → τ written out



τ elem. read from S during a phase

Es: M = 2 1, 8, 3, 2, 5, 0, 4, 6 3 runs

1, 3, 8

2, 5

0, 4, 6

1) phase: load M sorted elements in the min-heap H

2) item: writes to the output the minimum item within H, and load the next item from S in an ausiliary array U located in the memory, if next < current vmin; otherwise load next item in H if next > current min

3) end of a phase: H is empty and U consists of M sorted elements. At the end you only primary merge the runs, this save from the classical method of mergesort.

If i increase M to 2M i save 1 level.

$$\#\text{levels} = \log_2 \frac{N}{M} \longrightarrow \log_2 \frac{N}{2M} = \log_2 \frac{N}{M} - \log_2 2 = \log \frac{N}{M} - 1$$

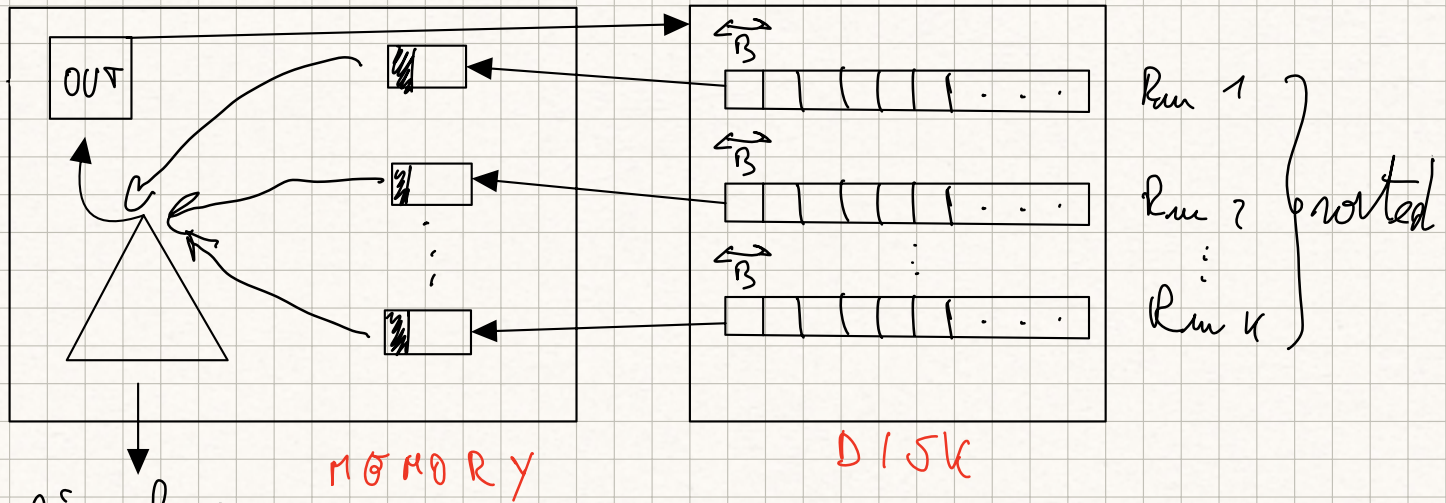
Binary Merge $= O\left(\frac{N}{B} \log_2 \frac{N}{M}\right) \neq O$

$\frac{M}{B}$

not fully exploiting the internal memory

→ only 3 blocks (pages) → 2 input
→ 1 output

Multi-way merge sort:



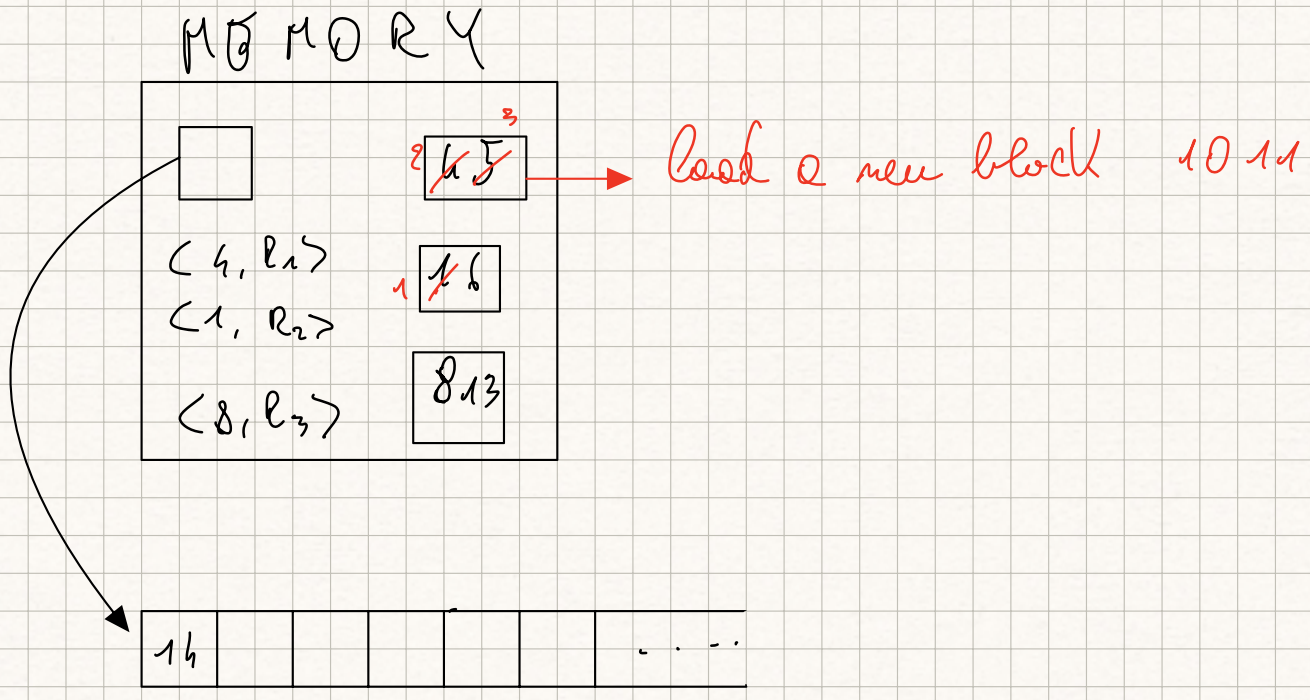
min-heap of size K → contains pair composed by $\langle$ min elem, num $\rangle$.

$O(\log_2 K)$

Es:

4 5	10 11	15 21	...	R_1
1 6	7 9	14 18	...	R_2
8 13	20 25	30 46	...	R_3

$B = 2$



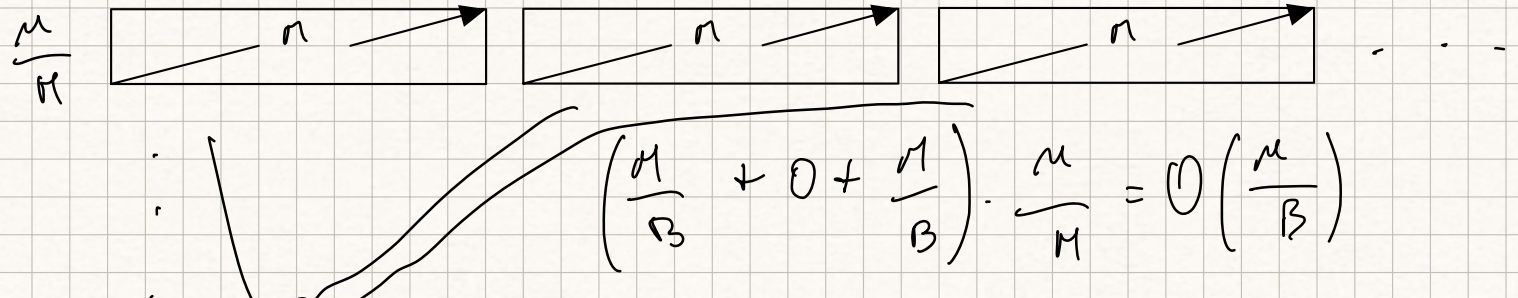
How can i put all the elements in the memory?

$(K+1)B + \underbrace{\text{heapsize}}_{K} \leq M$ K runs + B for merging

$(K+1)B + K \leq M$

$(K+1) \leq \frac{M}{B}$

$K < \frac{M}{B} - 1 = O\left(\frac{M}{B}\right)$ $\left(\frac{M}{B}\right) \rightarrow 2^{34}$ $\hookrightarrow 2^{15}$



$O\left(\frac{M}{B}\right)$

M/B

M/B

$O\left(\log_{\frac{M}{B}} \frac{N}{M}\right)$

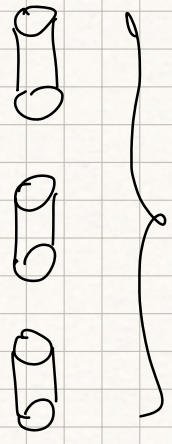
fan-out

total cost: $O\left(\cancel{\frac{N}{B}} + \frac{N}{B} \log_{\frac{M}{B}} \left(\frac{N}{M}\right)\right)$ I/Os

↳ initial number of runs

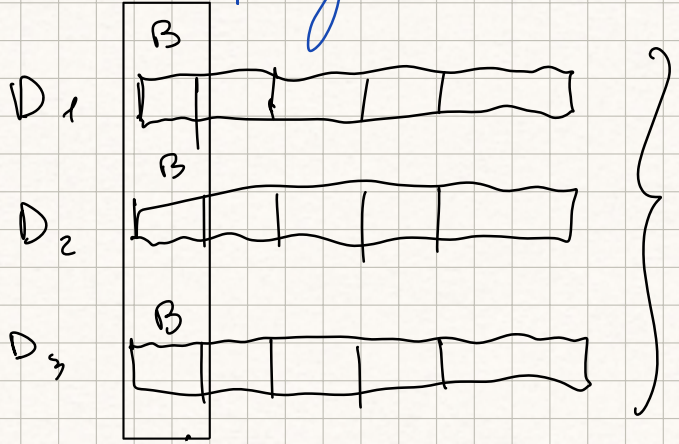
What if you have more than 1 disk? $D \geq 1$

parallelism: every disk can fetches one block



independent $\Longrightarrow$ 1 I/O $= 3B$ items
$= 3$ blocks

Disk Striping: look at D disks as one single disk with $B' = DB$



$B' = D \cdot B$

with $D = 1$ $O\left(\frac{N}{B} \log_{\frac{M}{B}} \frac{N}{M}\right)$ $D > 1$

↓ disk striping over D disks

$$O\left(\frac{N}{B'} \log_{\frac{M}{B'}} \frac{N}{M}\right) = O\left(\frac{N}{DB} \log_{\frac{M}{DB}} \frac{N}{M}\right) \text{ I/O s}$$

lower bound for sorting: $\Omega\left(\frac{N}{DB} \log_{\frac{M}{B}} \frac{N}{M}\right)$ I/O

slowing factor = $\frac{\text{disk striping: } \frac{N}{DB} \log_{\frac{M}{DB}} \frac{N}{M}}{\text{optimal alg: } \frac{N}{DB} \log_{\frac{M}{B}} \frac{N}{M}} =$

$$= \frac{\log_2 \frac{N}{M}}{\log_2 \frac{N}{DB}} \cdot \frac{\log_2 \frac{N}{B}}{\log_2 \frac{N}{M}} = \frac{\log_2 \frac{N}{B}}{\log_2 \frac{M}{B} - \log_2 D} = \frac{1}{1 - \log_{\frac{M}{B}} D} \geq 1$$

$\frac{N}{DB} \cdot \frac{1}{D}$

if $D \to \frac{M}{B}$ the **slowing factor** $\to +\infty$

Disk striping is less efficient as the number of disks increases.

lower bound for permuting 5.2.2. per la lode.

lower bound for sorting (D = 1)

every node is a comparison

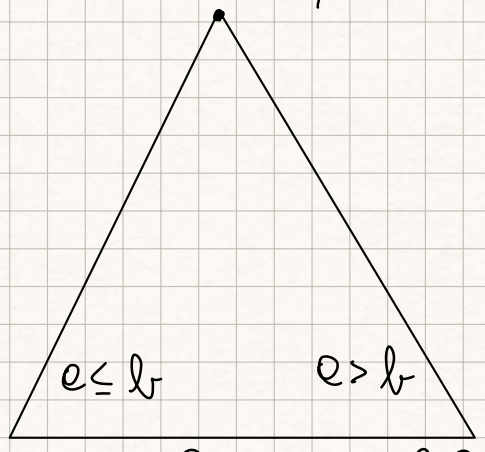
RAM only access

$2^h \geq l = n!$

$h \geq \log_2 n!$

$\log_2 n! \to n \log_2 n$

$h \geq n \log_2 n \Rightarrow h = \Omega(n \log n)$

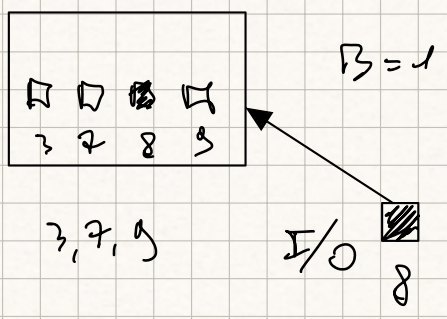


binary decision tree

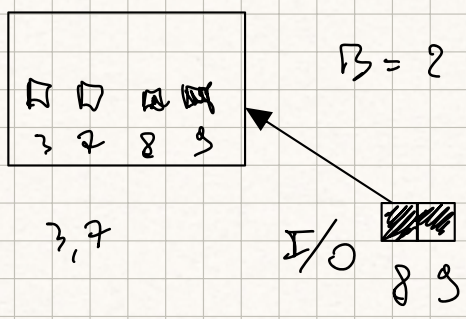
permutation

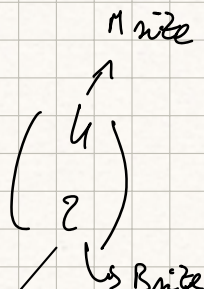
RAM/DISK ACCESS

M = 4



M = 4





# of possible allocation for B = 2

(1) $\binom{M}{B}$ # ways B items yet distributed among M-B items (sorted) already in memory

(2) first read → count permutation of B items

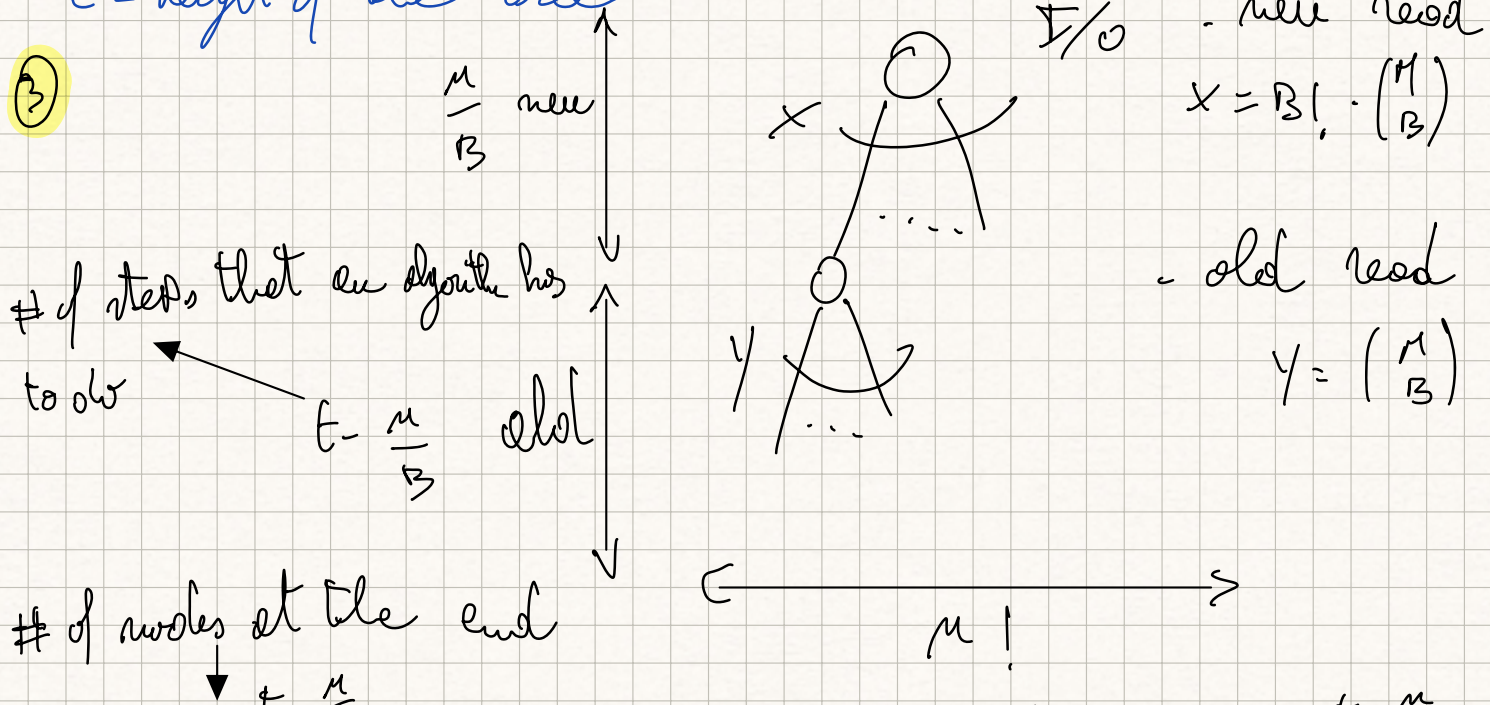
new read

read read

next read ⟶ no count

t = height of the tree

(3)



$$(B!)^{N/B} \cdot \binom{M}{B}^{t} \geq N!$$

property $\log_2 a! = a \log_2 a$

$\log_2 \binom{a}{b} = b \log_2 \frac{a}{b}$

$$\frac{N}{B} \log_2 (B!) + t \log_2 \binom{M}{B} \geq \log_2 N!$$

$$\frac{N}{B} B \log_2 B + t \cdot B \log_2 \frac{M}{B} \geq N \log_2 N$$

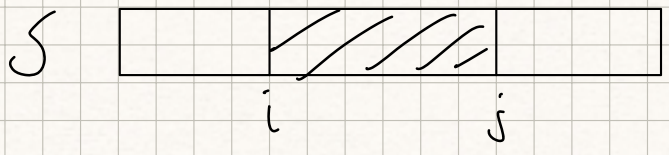
$$t \cdot B \log_2 \frac{M}{B} \geq N \log_2 \frac{N}{B} \longrightarrow t \geq \frac{N \log_2 N/B}{B \log_2 M/B} = \frac{N}{B} \cdot \log_{M/B} \frac{N}{B}$$

Merge based → Mergesort

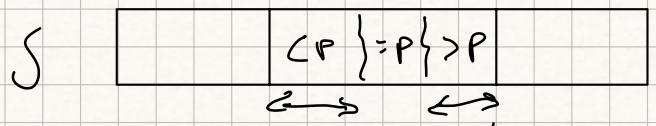
Distribution based → Quicksort

} Divide & conquer: Partition / divide; Recursion / conquer; Recombine

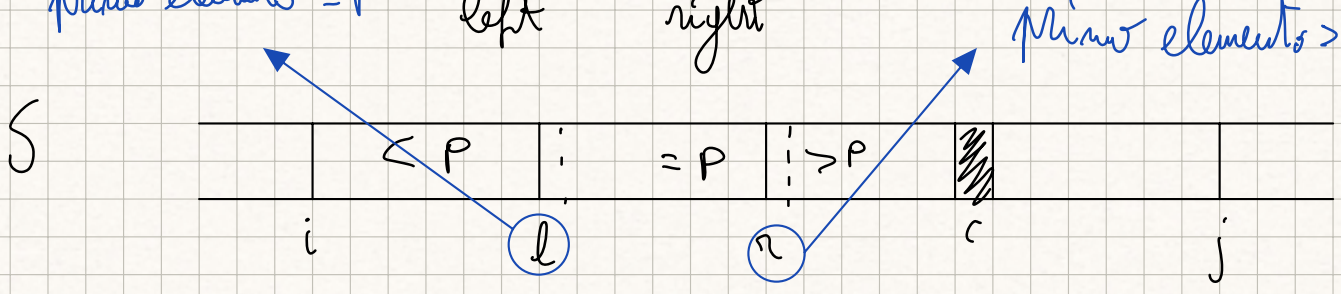
Quicksort: Partition (S, i, j)



p = PIVOT



three way partition



$S[c] = p$ Swap $S[c]$ with $S[r]$, increments r e c.

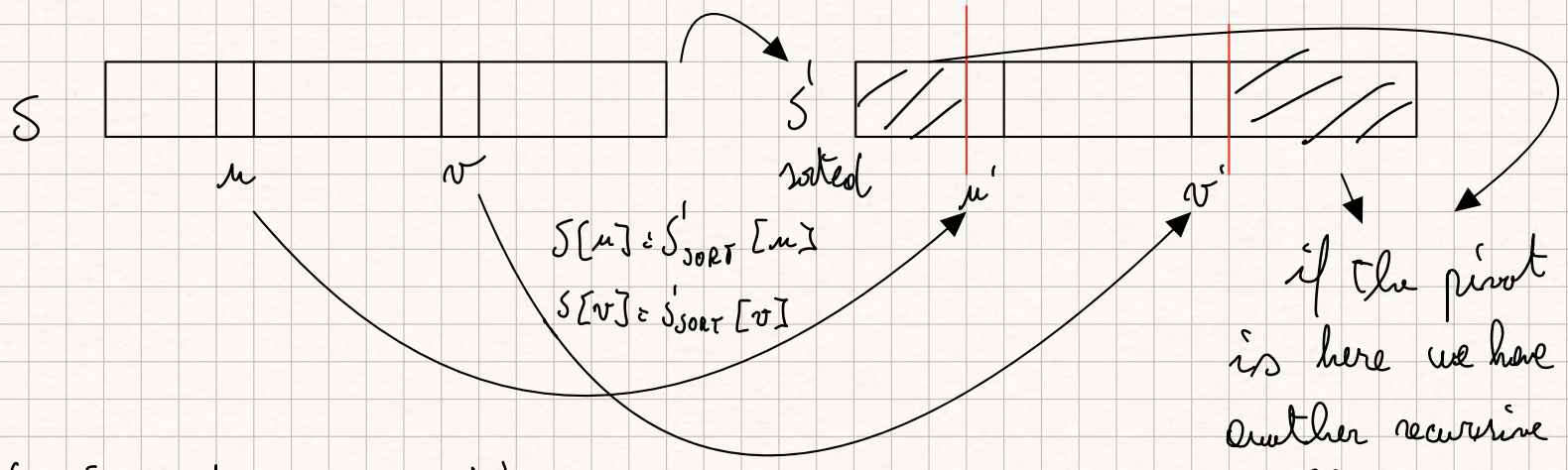
$S[c] < p$ Swap $S[c]$ with $S[l]$ & $S[c]$ with $S[r]$, inc. r, l, c

$S[c] > p$ increments c.

Expected time complexity of quicksort: $O(n \log n)$

Random binary $X_{u,v} = 1$ iff $S[u]$ compared with $S[v]$. One of them must be a pivot in order to be compared

$$\text{Exp \# comp.} = E\left[\sum_{u} \sum_{v>u} X_{u,v}\right] = \sum_{u} \sum_{v>u} E[X_{u,v}] = \sum_{u} \sum_{u>v} 1 \cdot P(X_{u,v}=1)$$



$P(u \text{ is pivot or } v \text{ is pivot})$

$$P(X_{u,v}=1) = \frac{2}{u'-v'+1} = \sum_{u'} \sum_{v'>u'} \frac{2}{u'-v'+1} \leq 2n \log n \quad \blacksquare$$

Bounded Quicksort (S, i, j):

while ($S - i > n_0$) // parameter that stop recursion because i will work only in local memory

    r = select position of pivot

    swap S[r] with S[i]

    p = Partition (S, i, j) // final position of the pivot

    if $p \leq \frac{i+j}{2}$ { BoundedQS (S, i, p-1); faccio la ricorsione sullo parte più corta.

        i = p+1; } // remove tail recursion

    else { BoundedQS (S, p+1, j)

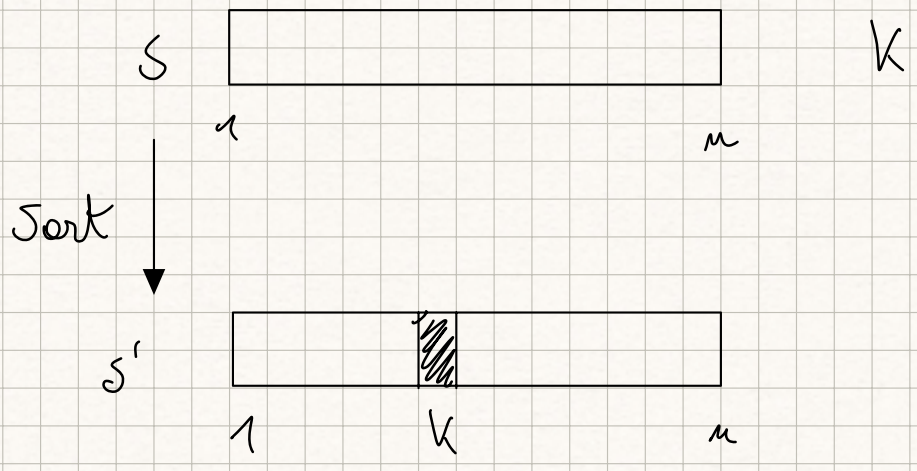
        j = p-1;

    }

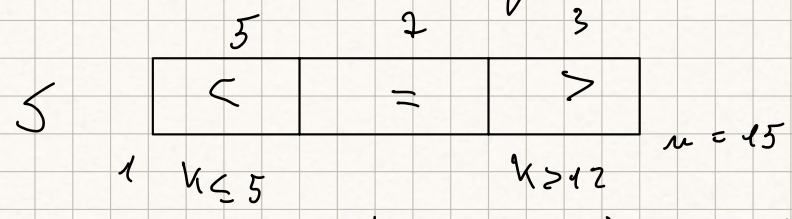
Insertion sort (S, i, j); // for very short arrays use IS.

↳ better than heapsort

Select k-th ranked element from S (unordered)



We use 3-way partitioning



$5 < k \leq 12$

↳ we do recursion $(n \cdot \log n)$

we do recursion $(n \cdot \log n)$

↓

all the elements are equal (n)

Recursive select (S, k): → k-th ranked item in S

r = random pos in $\{1, 2, \ldots, n\}$

$S_< = \{x \in S,\ x < S[r]\}$

$S_= = \{x \in S,\ x = S[r]\}$

$S_> = \{x \in S_1 \quad x > S[k]\}$

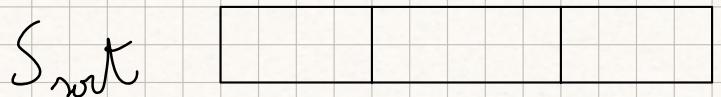
$n_< = |S_<|$

$n_= = |S_=|$

if $(k \le n_<)$ return RandSelect$(S_<, k)$

else if $(k \le n_< + n_=)$ return $S[k]$

else return RandSelect$(S_>, k - |S_<| - |S_=|)$



pivot should be here if chosen well

$|S_<| < \frac{2}{3}n$ $\quad$ $|S_>| < \frac{2}{3}n$

$T(n) = O(n) + p_< T(n_<) + p_> T(n_>)$

Time Complexity

rest of the array

$\hat{T}(n) \le O(n) + \frac{1}{3}\underbrace{\text{good}}_{\hat{T}(\frac{2}{3}n)} + \frac{2}{3}\underbrace{\text{bad}}_{\hat{T}(n-1)} \le O(n) + \frac{1}{3}\hat{T}(\frac{2}{3}n) + \frac{2}{3}\cdot\hat{T}(n)$

Prob.

worst case: when k-th element is 1° or last

$\hat{T}(n) - \frac{2}{3}\hat{T}(n) \le O(n) + \frac{1}{3}\hat{T}(\frac{2}{3}n)$

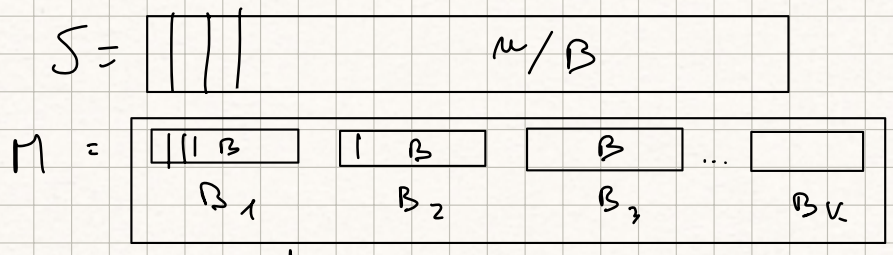
$\frac{1}{3}\hat{T}(n) \le O(n) + \frac{1}{3}\hat{T}(\frac{2}{3}n)$

per il master theorem $\hat{T}(n) = O(n)$

In terms of I/Os: $\hat{T}(n) \le O(n/B) + \hat{T}(2n/3) = O(n/B)$

increasing n the complexity can not reduce!

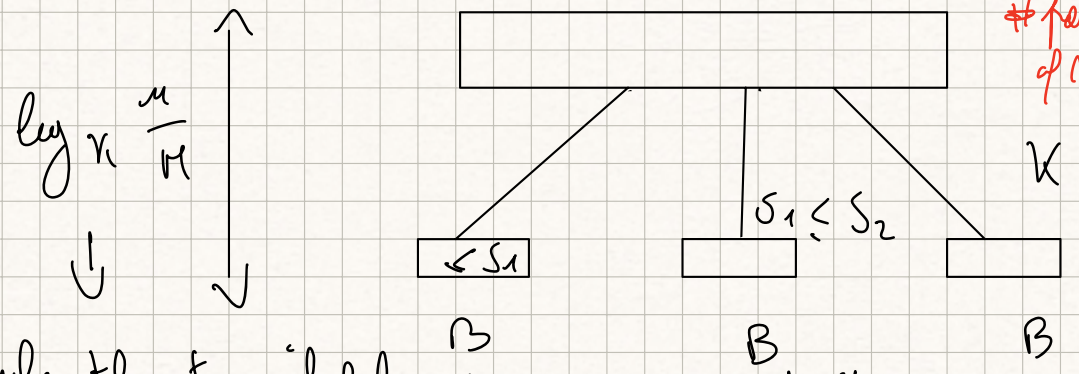
Multi-way Quicksort:



① more pivots $s_1, s_2, \dots, s_{k-1}$

$s_0 = -\infty \quad < \quad < \quad < \quad s_k = +\infty$

② $B_i = \{x \in S : s_{i-1} < x \leq s_i\}$ Buckets $\quad |B_i| = O\left(\frac{N}{k}\right)$

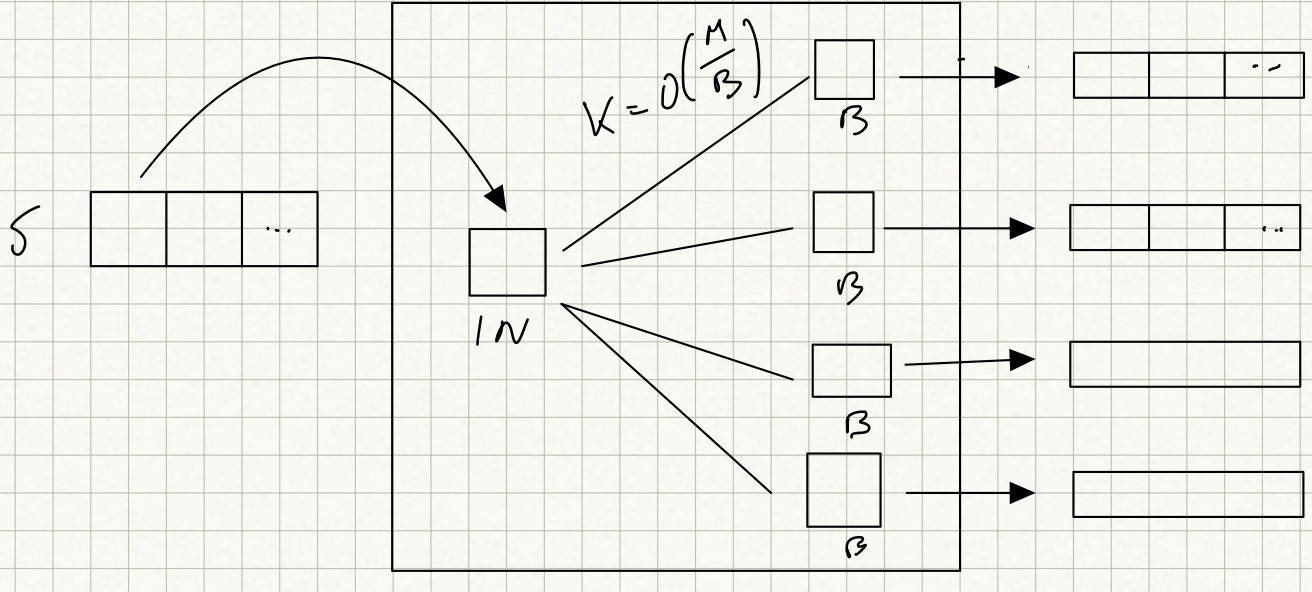


partitions of memory: $k = O\left(\frac{M}{B}\right)$

$k \quad O(N/B)$ I/O

when the tree is balanced # of partitioning phase $= O(\log_{\frac{M}{B}} \frac{N}{M})$

$k \cdot B \leq M \rightarrow k \leq \frac{M}{B}$



$O\left(\frac{N}{B} \cdot \log_{\frac{M}{B}}\left(\frac{N}{M}\right)\right)$ I/Os OPTIMAL

iff partition are balanced ($\sim \frac{N}{K}$ items each)

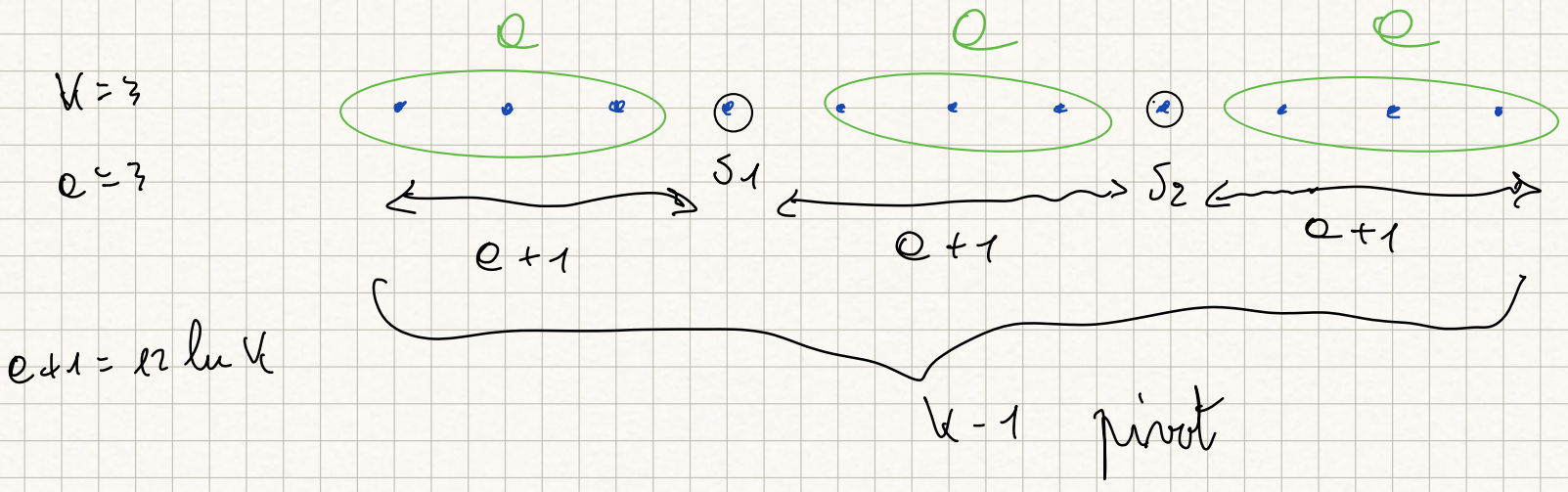
Oversampling (Multiway Quicksort):

1) Pick $(a+1)\cdot K - 1 = (a+1)(K-1) + a$ samples from $S \Rightarrow A$

2) Sort A

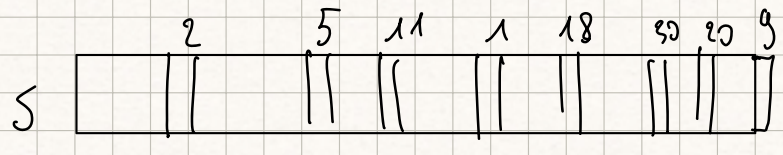
$a = O(\log K)$

3) Pick pivot: $S_i = A[(a+1)i]$ $i = 1, \dots, K-1$



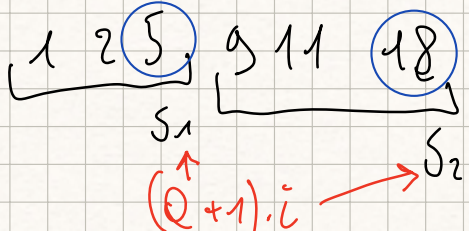
$B_i = \{ x \in S : S_{i-1} < x \leq S_i \}$ $S_0 = -\infty$ $S_K = +\infty$

Es: $K = 3$, $a = 2$



8 samples from S

$(a+1)(K-1) + a = (3 \cdot 2) + 2 = 8$



Th: Let $K \geq 2$ and $\alpha + 1 = 12 \ln K$, a sample of size $(\alpha+1)K - 1$ suffices to ensure that **all** buckets get $\leq \frac{4n}{K}$ items with prob. $\geq 1/2$

$$|B_i| \leq \frac{4n}{K} \qquad \forall i = 1, \dots, K$$

$n \longrightarrow$ $\begin{cases} O\left(\frac{n}{K}\right) \\ O\left(\frac{n}{K^2}\right) \\ \vdots \\ O(n) \end{cases}$ items × bucket

$\log_K \frac{n}{M}$

$K \cdot B \leq n$

$K \leq \frac{n}{B}$

$\frac{n}{K^i} \leq M$

Dim: $P\left(\exists B_i : |B_i| > \frac{4n}{K}\right) \leq 1/2$

$S_{SORT} =$ 

S_{i-1} B_i S_i

$\frac{K}{2}$ blocks → size of each block $= \frac{2n}{K}$

samples in $B_i = \alpha$

$\leq P(\exists t_j : t_j$ contains $< \alpha + 1$ samples)

$\leq \frac{K}{2} \cdot P(t_3 : t_3$ contains $< \alpha + 1$ samples) $\leq \frac{K}{2} \cdot \frac{1}{K} = \frac{1}{2}$

at the end

probability of 1 object

number of segment

① $P(\text{a sample occurs in } t_3) = \frac{2n/k}{n} = \frac{2}{k}$

② Fix $X = \#$ samples in $t_3 \longrightarrow E[X] = \frac{2}{k}\left[(a+1)\cdot k - 1\right] =$

$= 2(a+1) - \frac{2}{k} \geq 2(a+1) - 1 \geq \frac{3}{2}(a+1) \longrightarrow a+1 \leq \frac{2}{3}E[X]$

③ $P(X < a+1) \leq P\left(X < \left(1-\frac{1}{3}\right)E[X]\right) \leq$ (!!) $\left(1-\frac{1}{3}\right)E[X]$

Chernoff bounds $P(X < (1-\delta)E[X]) \leq e^{-\frac{\delta^2}{2}E[X]}$

$\delta = 1/3$

$\leq e^{-\frac{1}{9}\cdot\frac{1}{2}\cdot E[X]} \leq e^{-\frac{2}{12}\cdot\frac{1}{18}(a+1)} = e^{-\frac{1}{12}(a+1)} = e^{-\frac{1}{12}12\ln k}$

$= e^{-\ln k} = 1/k$

Plugging $1/k$ in $P(\exists B_i : |B_i| \geq 4n/k) \leq (k/2)\cdot(1/k) = 1/2$ ■

- Given a sequence of S items, $S = (i_1, i_2, \dots, i_n)$ and a positive integer $m \leq n$, the goal is to select m items from S uniformly at random ($P = m/n$)

- Rand(a, b) = returns a random number in $[a, b]$

- Assume that $m \leq \frac{n}{2}$

Dictionary = $\emptyset$

while $|D| < m$ do

P = rand $(1, n+1)$

if $P \notin D$ then insert (P, D)

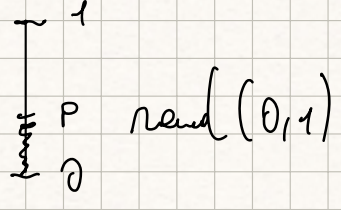
$P(\text{collision}) = \frac{(D)}{n} \leq \frac{m}{n} \leq \frac{1}{2}$

↳ take the same element

- Assume that n is known

take A with prob P

take B with prob $(1-P)$



$S = i_1\ i_2\ i_3\ i_4 \dots i_n$

$P = \frac{1}{n}\ \frac{1}{n-1}\ \frac{1}{n-2} \dots \frac{1}{1}$ $\quad m = 1$

$p = 1/n$

$P(\text{pick } i_j) = \frac{1}{n-j+1}$ $P(\text{pick } i_r,\ r=1\dots j-1) = \frac{1}{n}$

$P(\text{pick } i_1) = \frac{1}{n}$

$P(\text{pick } i_j \text{ given that } i_1,\dots,i_{j-1} \text{ have not been picked}) =$

$P(\text{pick } i_j) \cdot P(\text{not pick } i_1 \dots i_{j-1}) = \frac{1}{n-j+1} \cdot \left(1 - \frac{j-1}{n}\right) = \frac{1}{n-j+1} \cdot \frac{n-j+1}{n} =$

$= \frac{1}{n}$

if $n \geq 1$

$i_1, i_2, \dots, i_j$ — j = # selected items

$j=0$ $\frac{n}{n}$ $\frac{n-j}{n-1}$ $\frac{n-j}{n-j+1}$

$j=1$

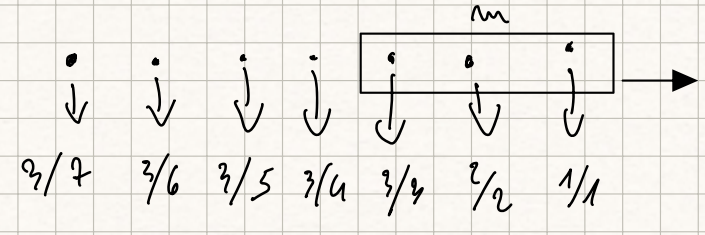
$p = \frac{n-j}{n-j+1} = \frac{n}{n} = 1$

$\frac{n-1}{n-1} = 1$

$\frac{n-2}{n-2} = 1$

Worst case: $\frac{1}{n/n}$ — n — $j=0$

Ex: $n = 3$, $n = 7$



I will surely pick this 3 element

$P(\text{pick } i_j) = \frac{1}{m-j+1}$ $\quad j = 1, \dots, m$ $\quad i_1, i_2, \dots i_m \rightarrow$

↑ Known

1) select the indexes $\{3, 7, 13\}$

2) pick items $\min\left\{m, \frac{n}{B}\right\}$

if n unknown: $P(\text{pick } i_j) = \frac{m}{n}$ $\quad \forall j$ to be guaranteed

0,1 0,7 0,55 0,05 0,9 0,09

$i_1, i_2, i_3, i_4, i_5, i_6$ $\rightarrow$ random number $0 < r_j < 1$

We define $\forall i_j \Rightarrow < r_{j_m}, j >$ Rand (0,1) to get probability

H = heap including the m priors of minimum priority (r_j)

es: $P(\text{pick } i_j)$

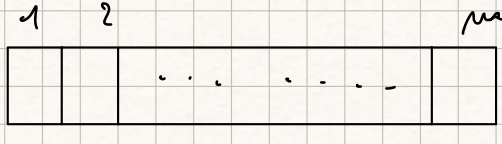
0.1 0.7 0,55 0,05 0,9 0,09

$i_1, i_2, i_3, i_4, i_5, i_6 \dots$

H: $< 0,1 , 1 >$

$< 0,7 , 2 >$ ~~$<$~~ $< 0,55, 3 >$ gets in H because $0,55 < 0,7$ $\quad < 0,1 , 1 >$ $\quad$ ~~$< 0,55, 3 >$~~ $\quad < 0,05 , 4 >$

Reservoir sampling: $R =$

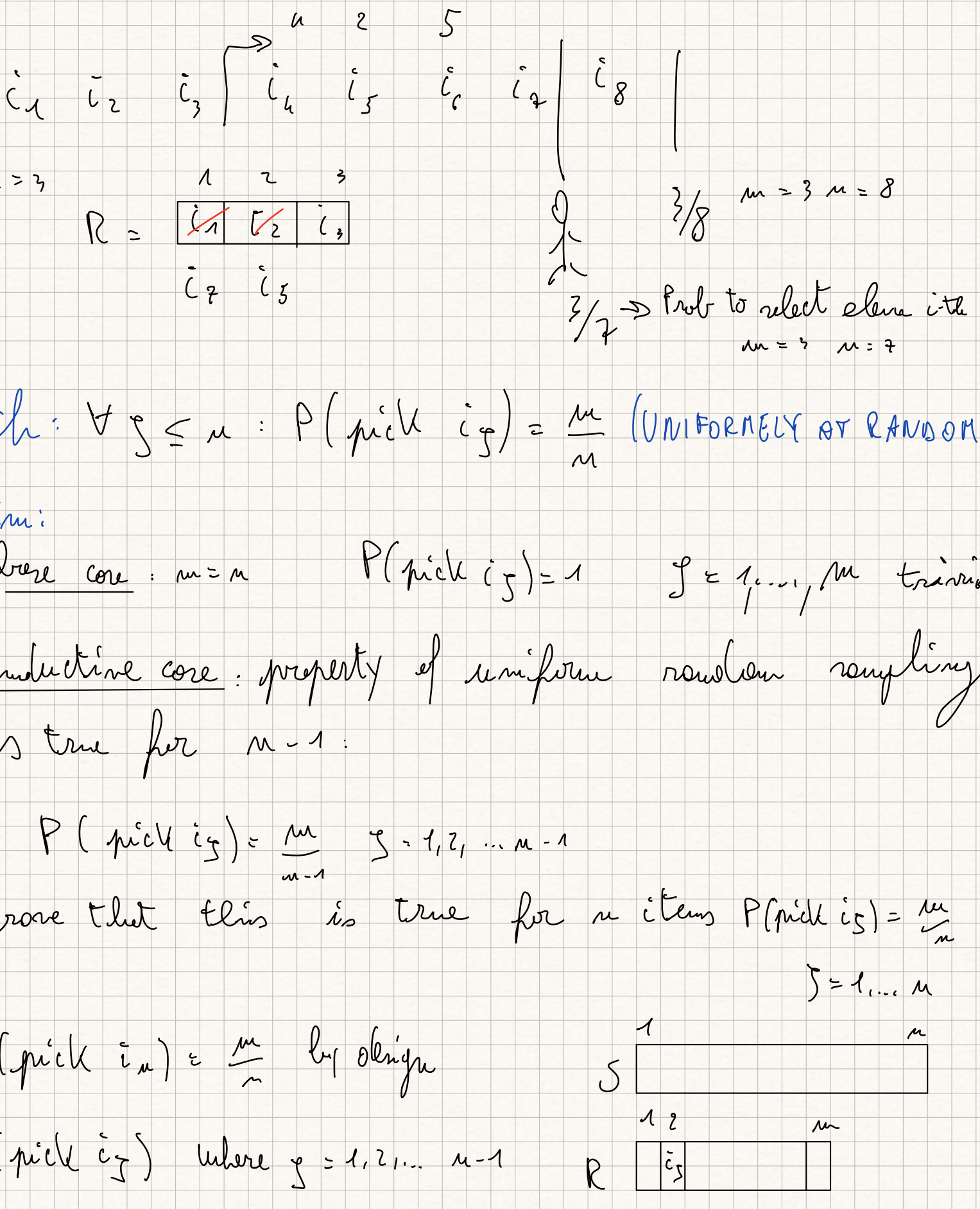


1) initialize $R = i_1, i_2, \dots, i_m$

2) For $j = m+1$

$h = \text{rand}(1, j) \in N$

if $(h \leq m)$ $R[h] = i_j$



Th: $\forall j \leq n : P(\text{pick } i_j) = \frac{m}{n}$ (UNIFORMELY AT RANDOM)

Dim:

base case : $m = n$ $\quad P(\text{pick } i_j) = 1 \quad j = 1, \dots, m$ trivial

inductive case : property of uniform random sampling is true for $n-1$:

$$P(\text{pick } i_j) = \frac{m}{n-1} \quad j = 1, 2, \dots n-1$$

Prove that this is true for n items $P(\text{pick } i_j) = \frac{m}{n}$

$j = 1, \dots n$

$P(\text{pick } i_n) = \frac{m}{n}$ by design

$P(\text{pick } i_j)$ where $j = 1, 2, \dots n-1$

$P(i_j$ is in R after $n-1$ steps AND ($(i_n$ is not picked) OR (i_n is picked but stored not in position of i_j)))

ind hyp: $P(i_j \text{ picked}) = \frac{m}{n-1}$ for $j = 1, 2, \dots, n-1$

$P(i_j \text{ is in } R \text{ after } n-1 \text{ steps}) \cdot P((i_n \text{ not picked}) \text{ OR } (i_n \text{ is picked AND not stored where is } i_j))$

$$\frac{m}{n-1} \cdot \left[\left(1 - \frac{m}{n}\right) + \left(\frac{m}{n} \cdot \frac{m-1}{m} \right) \right] =$$

$$= \frac{m}{n-1} \cdot \left(\frac{n-m}{n} + \frac{m-1}{n} \right) = \frac{m}{n-1} \cdot \left(\frac{n-1}{n} \right) = \frac{m}{n} \quad \blacksquare$$

PROBLEM: we need to generate too much random numbers.

e.g.: $m = 2$ $n = 8$ ← sequence length

$$P \leq \frac{m-j}{n-j+1}$$

1	2	3	4	5	6	7	8 ← j
a	b	c	d	e	f	g	h

.5	.5	0,1	.5	1	0,02	1	1

R = | | |

$j=0$ $j=1$: $\frac{m-j}{n-j+1} = \frac{2-0}{8-1+1} = \frac{2}{8} = 0,25$ ✗

$j=0$ $j=2$: $\frac{2-0}{8-2+1} = \frac{2}{7} \sim \frac{1}{2}$ ✗

$S=0 \quad J=3 : \dfrac{2-0}{8-3+1} = \dfrac{2}{6} = \dfrac{1}{3} \sim 0,1$ ✓

$S=1 \quad J=4 : \dfrac{2-1}{8-4+1}$

$S = (a, b, c, d, e, f, g, h, i, \dots) \qquad m = 3$

$h = (\cdot \; \cdot \; \cdot \; 2 \; 4 \; 1 \; 2 \; 3 \; 1)$ ← status R

	1	2	3
R =	~~a~~	~~b~~	~~c~~
	~~f~~	~~d~~	h
	i	g	

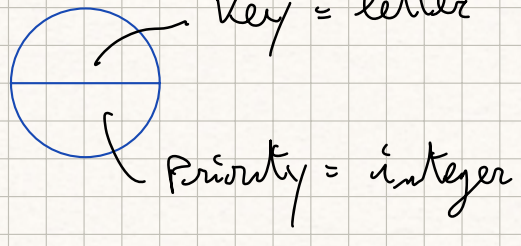
heap

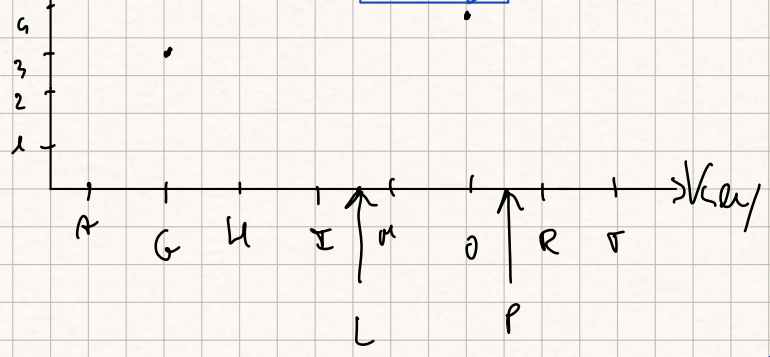
TREAP

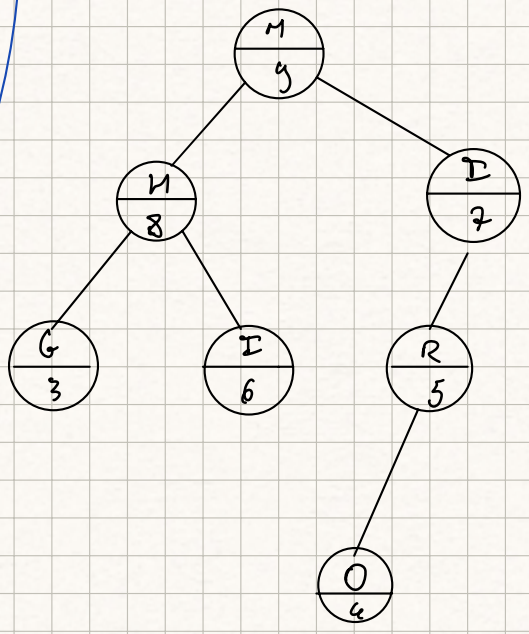
Tree

min

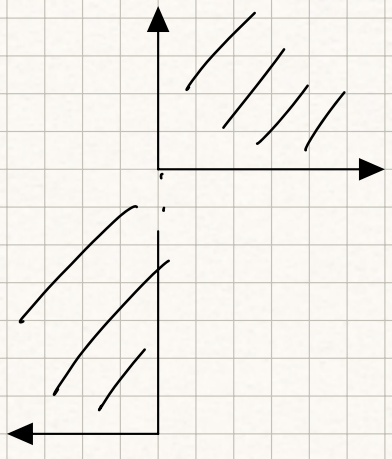
max



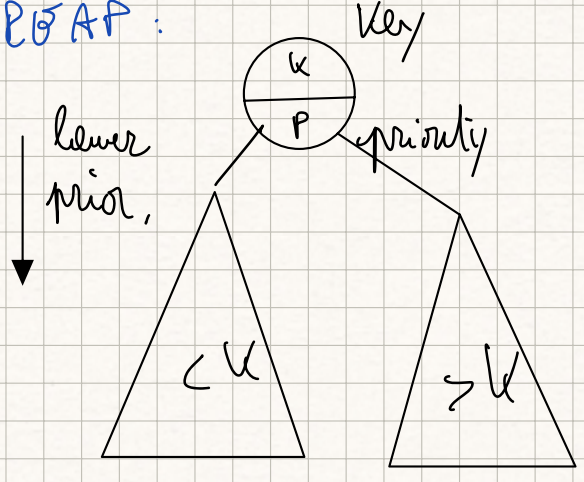




3 sided range query



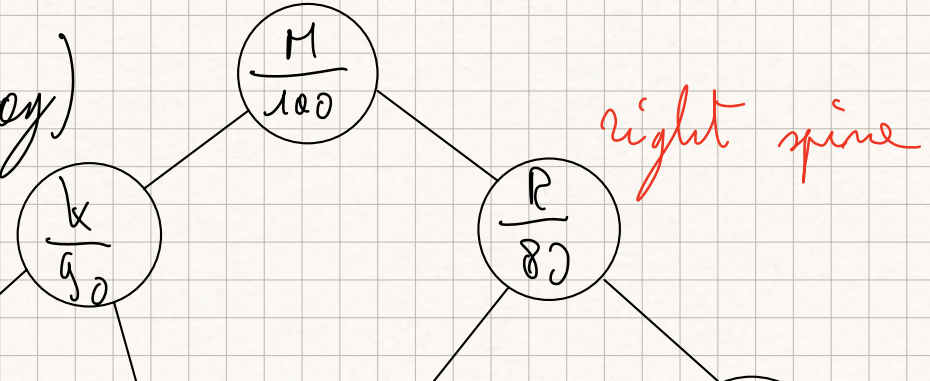
TREAP:

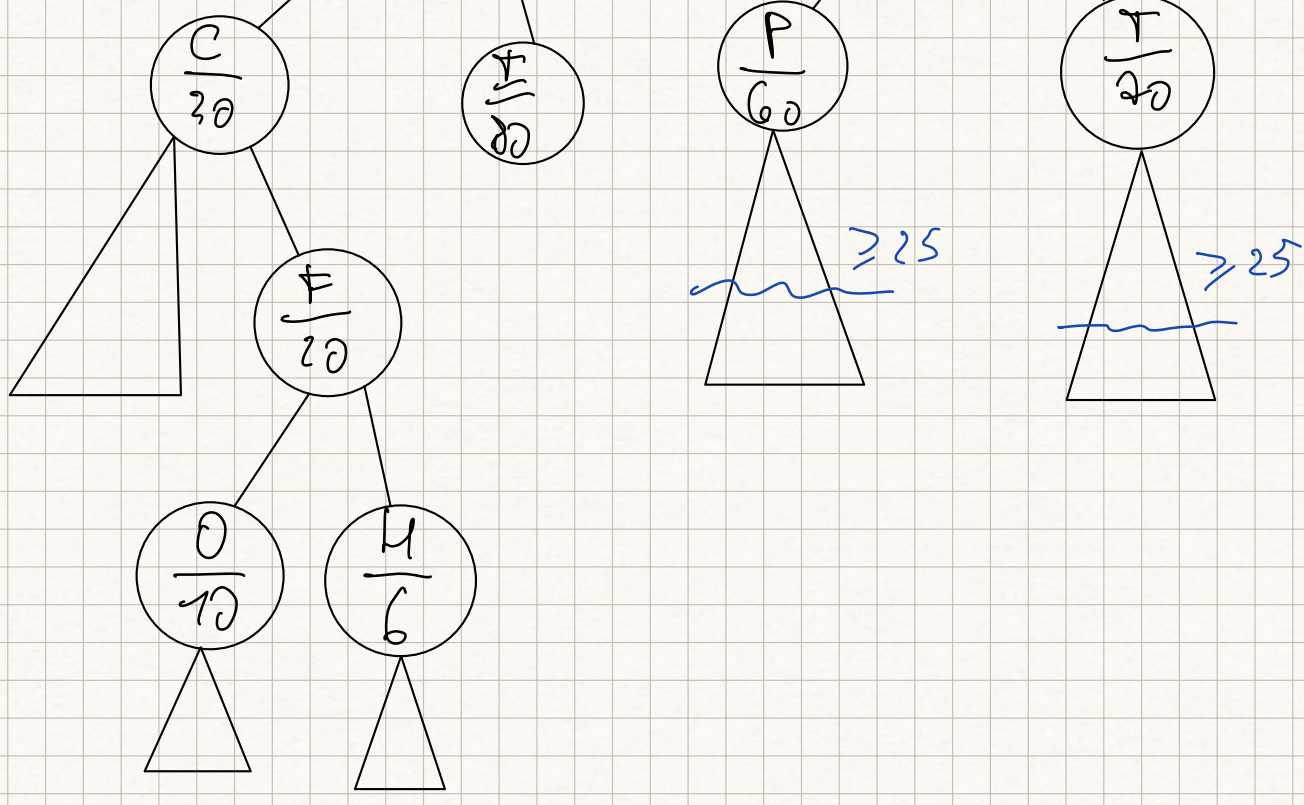


- Binary search tree on keys
- Heap (max) on priorities

$\underbrace{[6,7]}_{x\text{-axis}} \times \underbrace{[25,+\infty)}_{y\text{-axis}}$

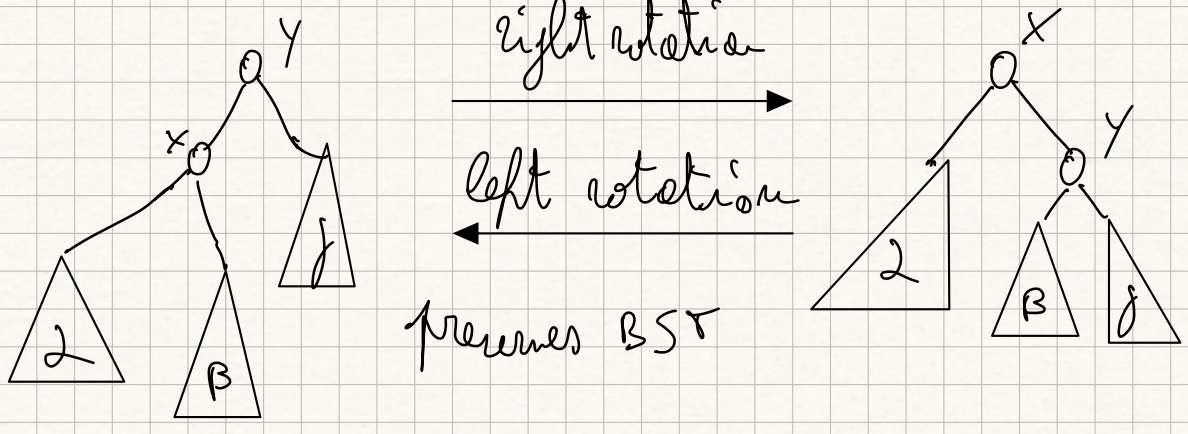
$O(h)$ = traversal (zig-zag)





$O(h) +$ { # returned answers ; # visited but stopped nodes }

$\leq$ 2 · # returned answers



preserves BST

$x \subseteq y$

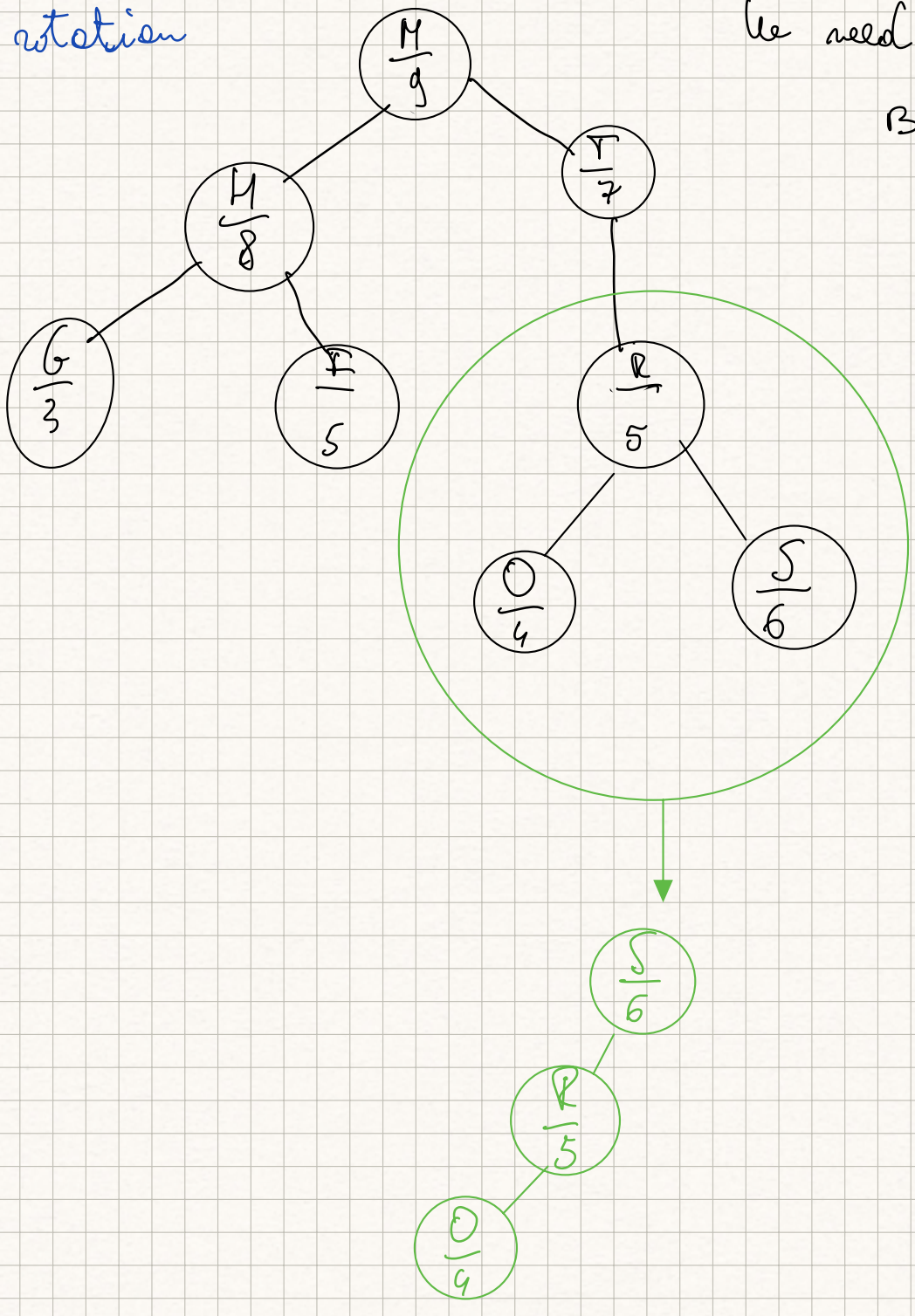
$$\frac{\text{promotes } x}{\text{demotes } y}$$

Search(K) search in a BST (binary search tree)

Insert(K, P): O(h) (K/P) → T

1) Search
2) rotation

We need to preserv the BST & heap

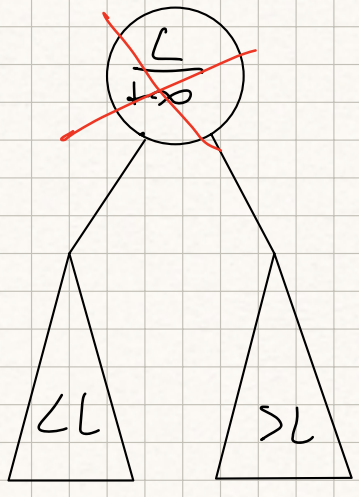


Delete (x) : $O(h)$ 1) search 2) set $-\infty$ priority to node to del 3) rotate until it has no childs

Split (x) : $O(h)$

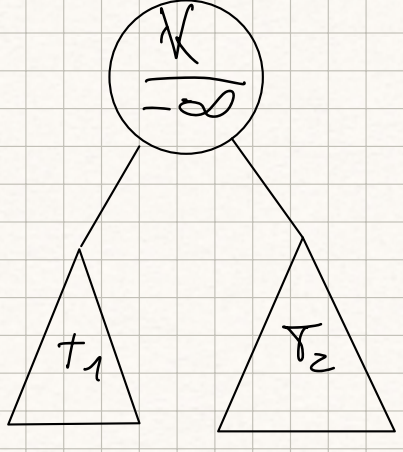
split $(L) \rightarrow T_{<L}, T_{\geq L}$

- Insert $(L, +\infty)$



Merge (T_1, T_2) : $O(h)$

$\text{keys}(T_1) < \text{keys}(T_2)$



$T_1 < x < T_2$

rotate $\left(\frac{x}{-\infty}\right)$ to the root until it has no childs

Split $(k) = \langle T_1, T_2 \rangle$ $\quad O(h)$

$\leq k \quad > k$

Insert a dummy node with k as key and $+\infty$ priority and rotate it to the radix and then delete it

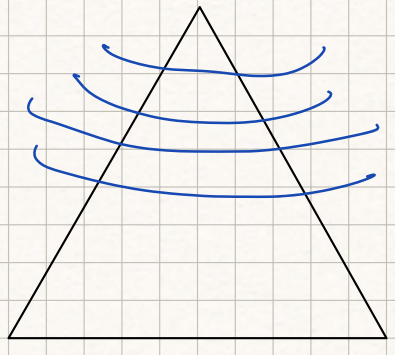
keys priority

3-side query $= Q : [a, b] \times [c, +\infty)$

Split (e) $\begin{cases} T \leq e & (-\infty, e] \\ T > e & (e, +\infty) \end{cases}$

Split $(\emptyset, T > e)$ $\begin{cases} [e, b] \\ (b, +\infty) \end{cases}$

top-bottom approach



es: $[c, d] \times [e, +\infty)$ on first treap

Given a dictionary D, build a BST balanced

- AVL-tree, 2-3 trees, Red(Black trees, ... h: $O(\log n)$

Th: Build a treap in which keys $\in D$, priorities are random numbers. height = $O(\log n)$ on expectation

Dim:

$$A_k^i = \begin{cases} 1 & \text{if key } x_i \text{ is a proper ancestor of key } x_k \\ 0 & \text{otherwise.} \end{cases}$$

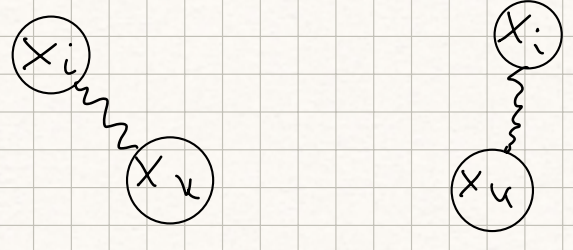
random indicator variable

$D = \{x_1, x_2, \dots, x_n\}$ with priorities $p_1, p_2, \dots, p_n$

$$\text{depth}(x_k) = \sum_{i=1}^{n} A_k^i \quad \text{\# of ancestor of } x_k$$

$$\mathbb{E}[\text{depth}(x_k)] = \mathbb{E}\left[\sum_{i=1}^{n} A_k^i\right] = \sum_{i=1}^{n} \mathbb{E}[A_k^i] =$$

$$= \sum_{i=1}^{n} P(x_i \text{ is a proper ancestor of } x_k) = \sum_{i} 1 \cdot P(A_k^i = 1)$$



max heap: $\text{prior}(x_i) > \text{prior}(x_k)$

prove heap in the notes

1) $X(i,k) = \{x_i, x_{i+1}, \dots, x_k\}$ $i < k$ #items $= k - i + 1$

$x_1 \dots x_{i-1}$ ← ↳ → $x_{k+1} \dots x_m$

2) $X(k,i) = \{x_k, x_{k+1}, \dots, x_i \mid i > k$ #items $= i - k + 1$

$$\mathbb{E}[\text{depth}(x_m)] = \sum_{i=1}^{m} P\left(\text{X1} \rightsquigarrow \text{Xi}\right) = \sum_{i<k}^{k-1} \frac{1}{k-i+1} + \sum_{i>k}^{m} \frac{1}{i-k+1} <$$

$i=1 \to 1/k$ ⋮ $i=k+1 \to 1/2$

$i=2 \to 1/k-1$ ⋮ $i=k+2 \to 1/3$

⋮ ⋮ ⋮

$i=k-1 \to 1/2$ ⋮ $i=m \to \frac{1}{m-k+1}$

$\ln m \cong \left(\frac{1}{2} + \frac{1}{3} + \frac{1}{4} \dots + \frac{1}{m}\right) H_k$

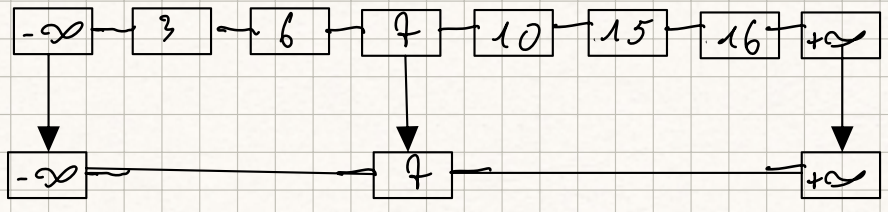
$\ln \geq \ln k$

$$< \ln k + \ln(m-k+1) - 2 < 2 \ln m - 2 \to O(\log m)$$

Skip list '90

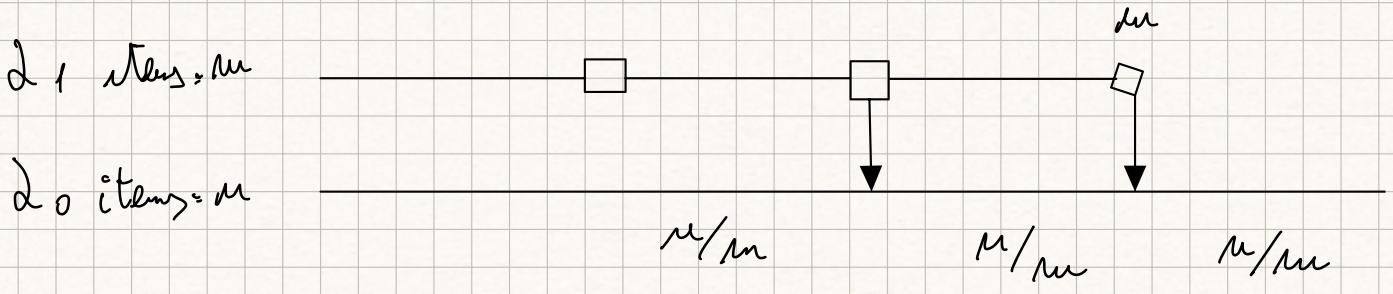
- Level DB
- REDIS (SortedSet)
- Rocks DB
- HBase

L_0 list



L_1

n = # items = $|L_0|$



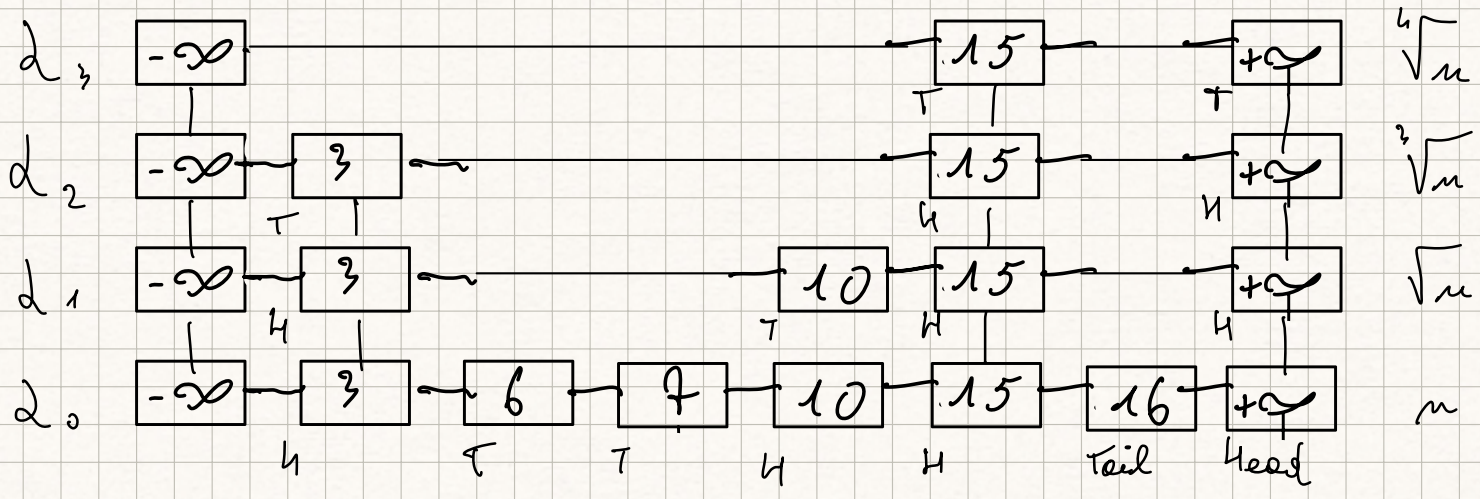
cost of a search $O\left(n + \frac{n}{n}\right)$

if $n = \frac{n}{n}$ minimizing the search cost

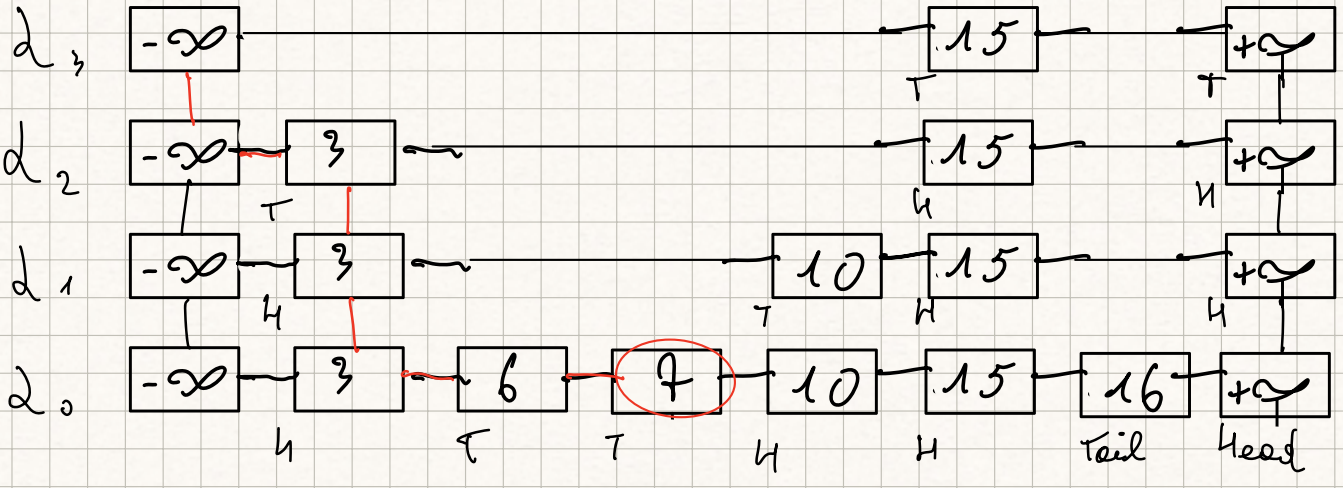
$n^2 = n$

$n = \sqrt{n}$ optimal choice

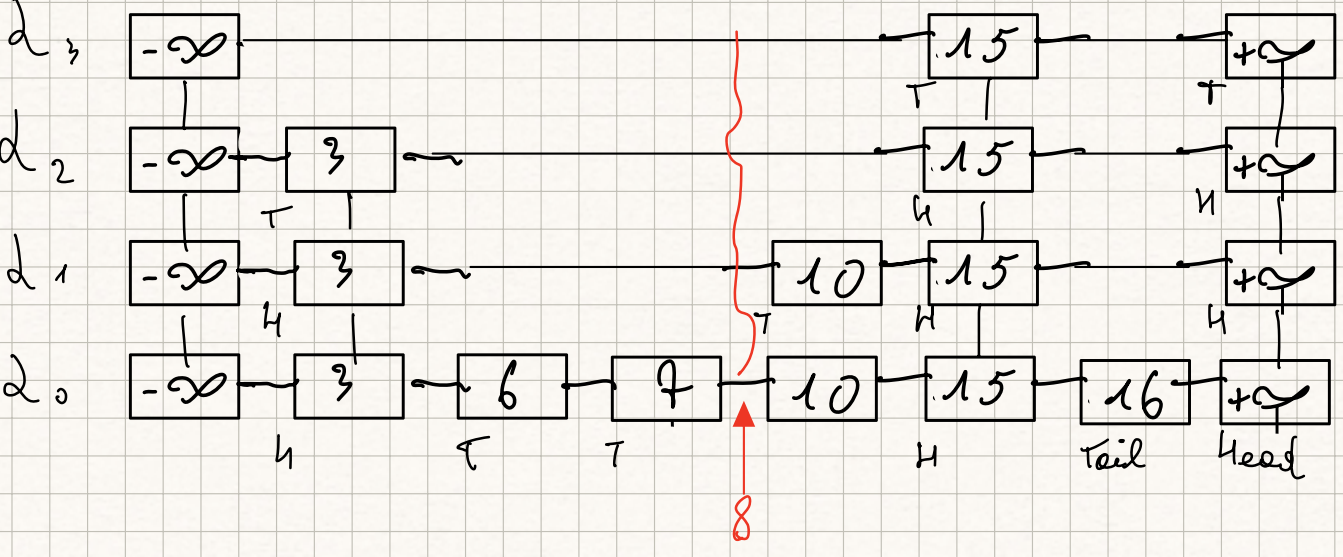
use a randomized approach $E[\text{height}] = O(\log n)$



Search 7 in Skip list



Insertion of 8 in Skip list



Start tossing a coin and decide in which list put it

Time complexity of search = O(# vertical links traversed + # horizontal links traversed)

height

level = $\max_k L(k)$

level for key k

$P(L(k) \geq l)$ = at least l heads = $\left(\frac{1}{2}\right)^l$

$P(L \geq l) = P(\max_k L(k) \geq l) = P(\max\{L(1), L(2), \ldots, L(n)\} \geq l)$

union bound $\rightarrow \leq \sum_{k=1}^{n} P(L(k) \geq l) = \sum_{k=1}^{n} \left(\frac{1}{2}\right)^l = \frac{n}{2^l}$

Probability that height is greater or eq. than l

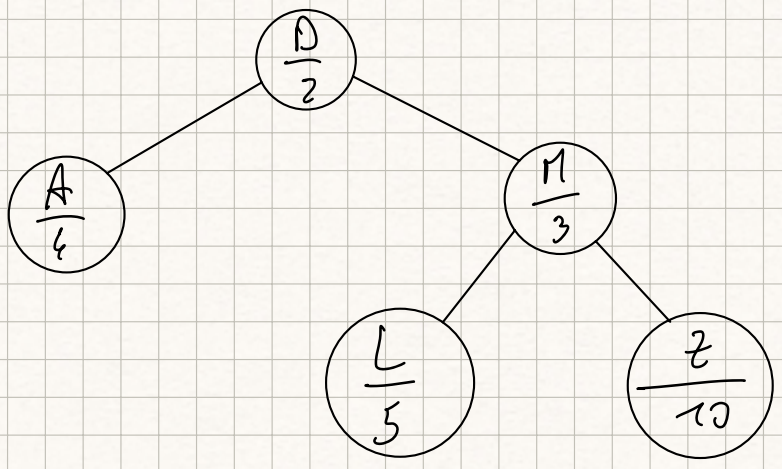
if $l \leq \log_2 n \rightarrow \frac{n}{2^l} \geq 1$ Trivial Prob. doesn't hold because ≥ 1

if $l = c \cdot \log_2 n \rightarrow c > 1$

$P(L \geq l) = \frac{n}{2^{c \log_2 n}} = \frac{n}{n^c} = \frac{1}{n^{c-1}}$ $c > 1$ with high probability

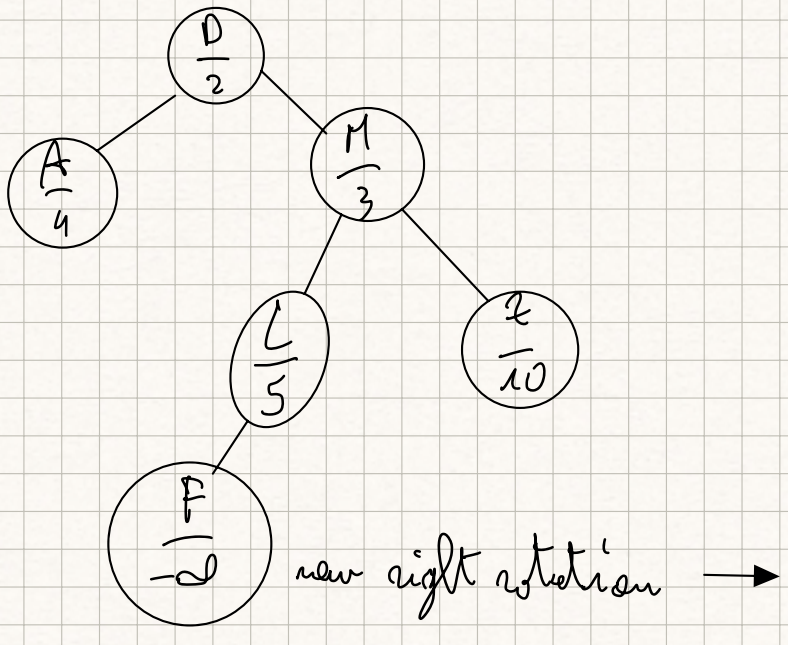
We conclude that with high probability, a skip list has O(log n) levels.

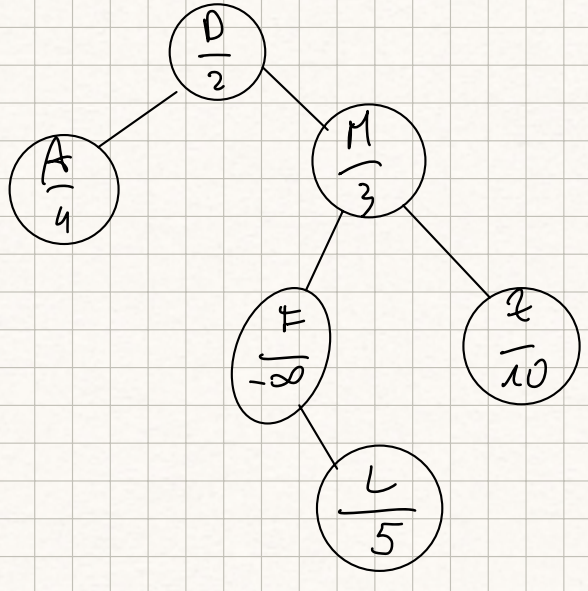
ls: TREAP(min)

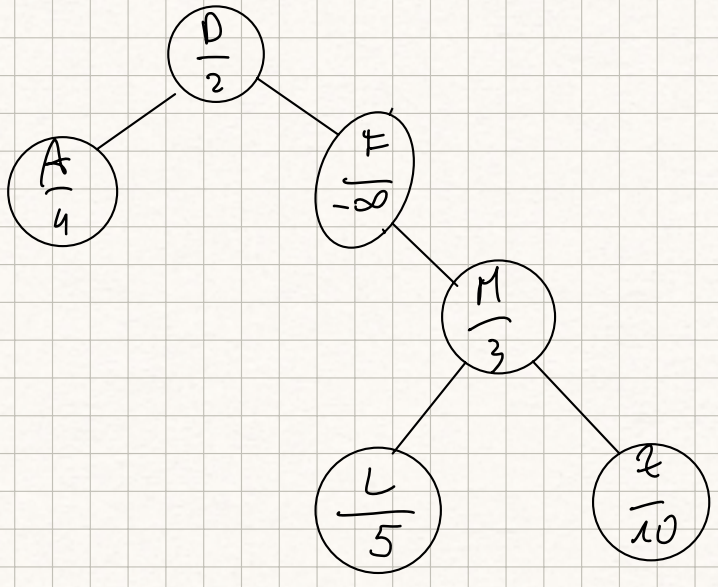


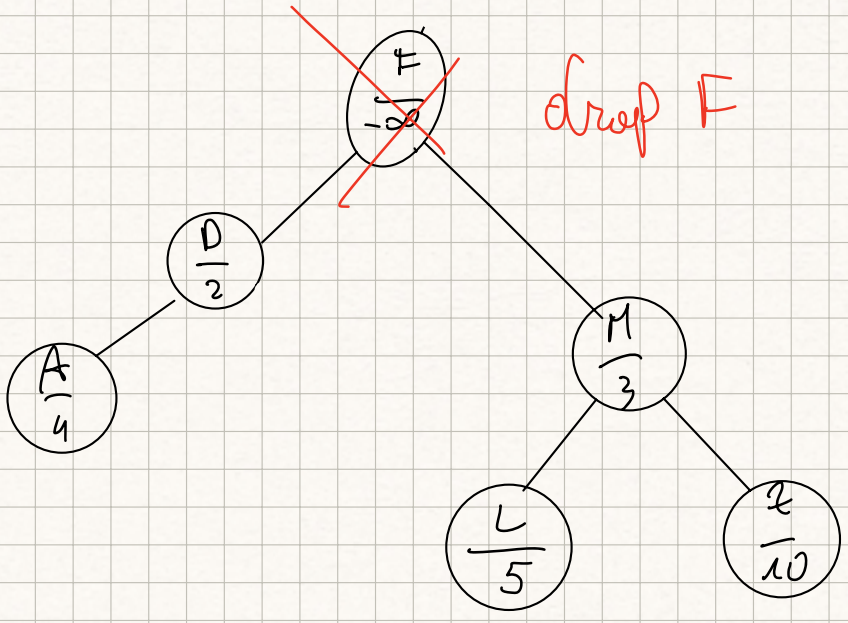
Split (F)

- Insert (F, -∞)

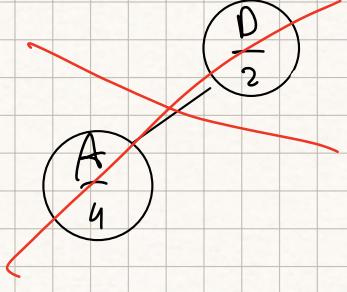


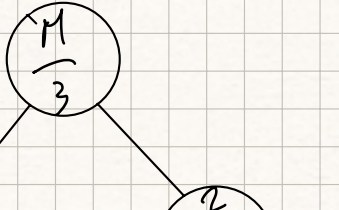


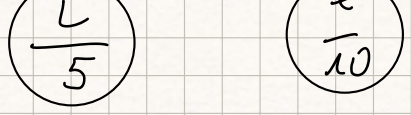




3- niveau question $[F, \infty] \times [G, -\infty]$

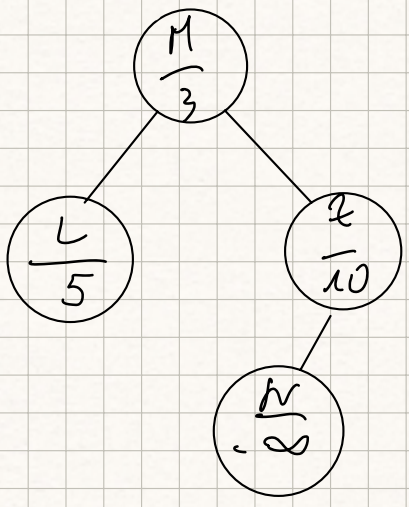


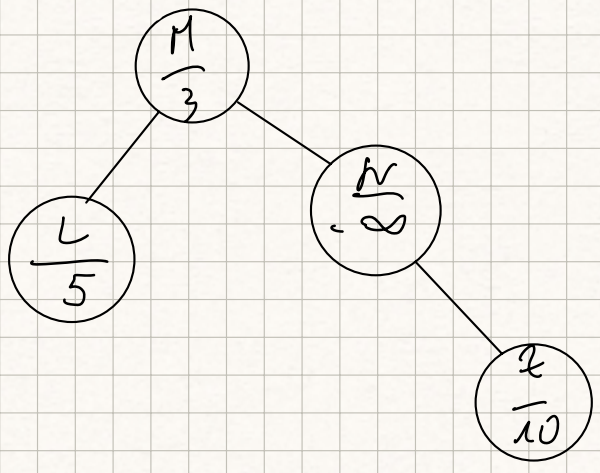


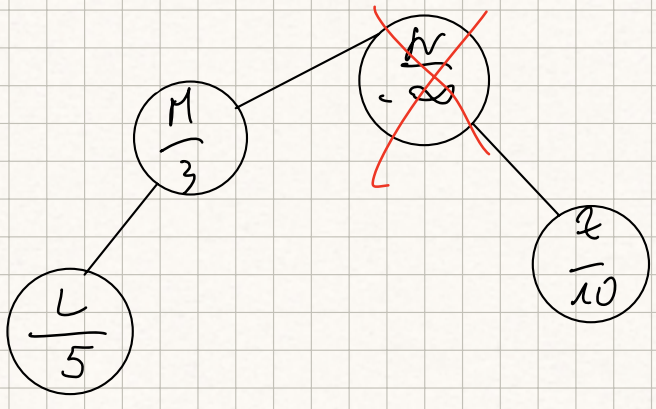


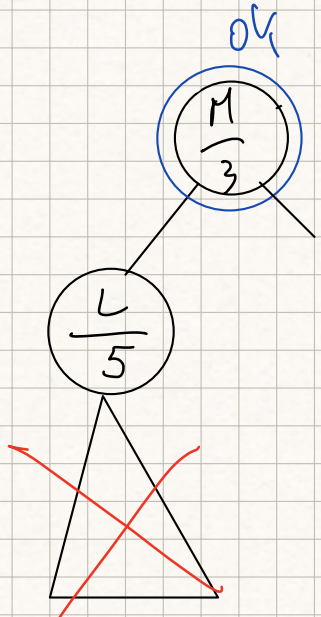
1 Split F (already done)

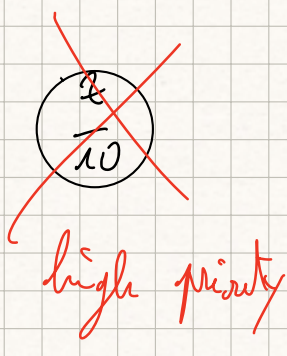
2 Drop left subtree (item < F)











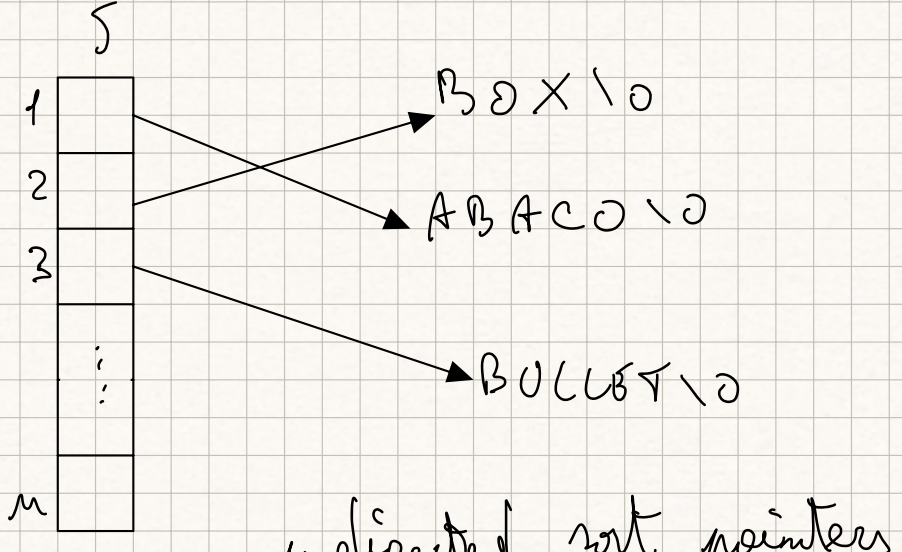
String sorting (RAM):

$S[1, n]$ = array of pointers to strings

N = Tot length of strings

σ = alphabet size

$L = \frac{N}{n}$ average string length



indirect sort pointers

qsort (s, n, char *, cmp)

$O(n \log n)$ on expectation

$O((n \cdot \log n) \cdot l)$ time complexity

→ avg string len = $\frac{N}{n}$ ← Tot string len
→ # of string

$O(n \log n)$ cache misses / I/Os

distinguishing prefix of a string in a set d_S

$$S \begin{cases} s_1 = abaco & d(s_1) = 4 \\ s_2 = abate & d(s_2) = 4 \\ s_3 = abb & d(s_3) = 3 \end{cases} \quad d = \sum_s d_s = 11$$

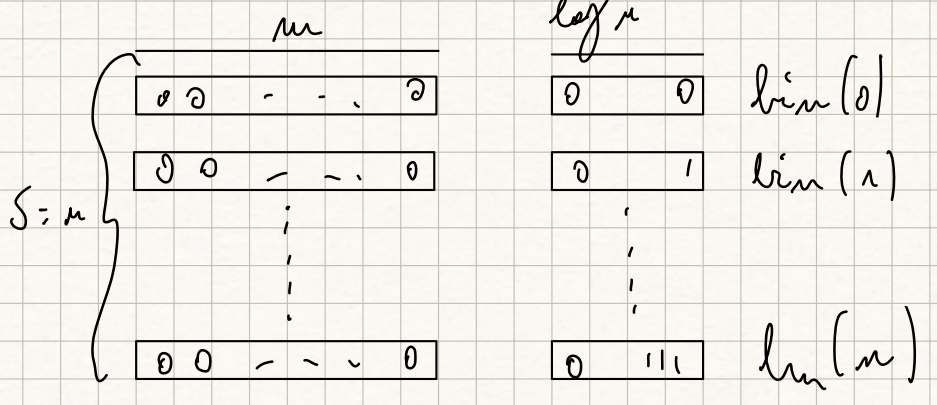
a) One notes has to look at the distinguishing prefix of $s \in S$

$\Omega(d)$

b) Sorting S needs to sort by the first char of a string $\Omega(n \log n)$

Lower bound # comparisons to sort n strings $\Rightarrow$

$= \Omega(d + n \log n)$



$\forall s : d_s > n \Rightarrow d = \Omega(n \cdot n)$

lower bound on that $S = \Omega(n \cdot n + n \log n)$ char emp

length of the strings we are comparing.

$$q\text{sort} = O((n \log n) \cdot (n + \log n))$$

n = constant $= O(1)$ lower bound $= \Omega(n \log n)$ qsort $= O(n \log^2 n)$

$n > \log n$ l. b. $= \Omega(n \cdot n)$ qsort $= O(n \cdot n \log n)$

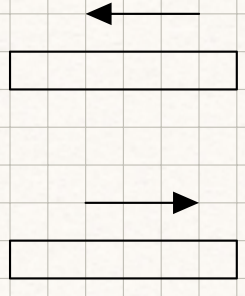
$n \leq \log n$ lower bound $= \Omega(n \log n)$ qsort $= O(n \log^2 n)$

Q sort is not optimal. There is a factor scale of $O(\log n)$ between lower bound and qsort

Radix-based sort:

- LSD: least-significant digit
- MSD: most-significant digit

} first



FRIOT

If alg. will introduce a data structure

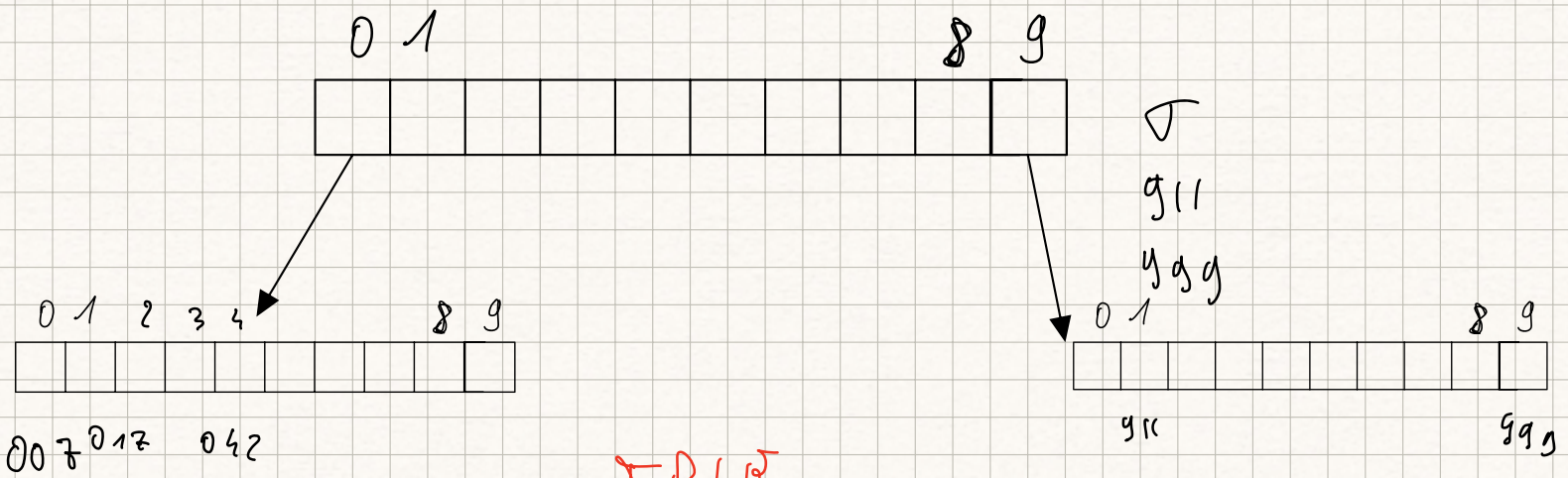
σ = distinguishing prefix $\sigma = |\Sigma|$ alphabet size

n = # strings

N = Total length of the strings

MSD

$S = \{007, 017, 042, 911, 999\}$



TRIE

size trie = # nodes $= O\left(\sum_s d_s\right) = O(d)$

leaves = n = # strings

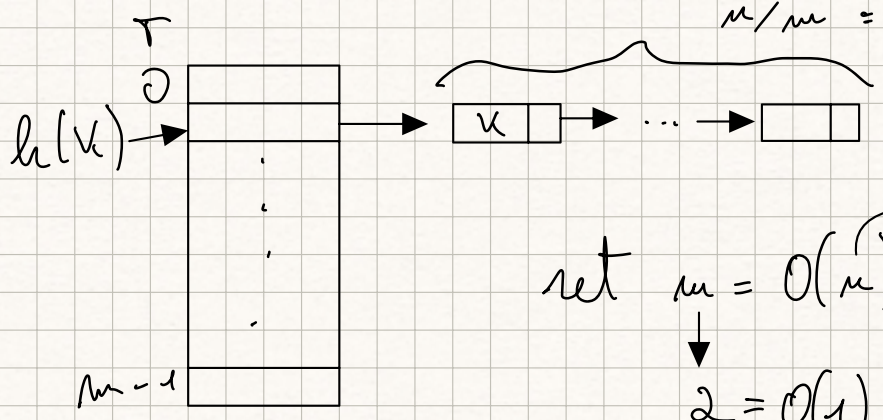
space $= O(\#\text{nodes} \cdot \sigma) = O(d \cdot \sigma)$ $\sigma = 2^{64}$ a lot of space!

time construct trie $= O(\underbrace{d}_{\text{string distr.}} + \underbrace{d \cdot \sigma}_{\text{node creation}} + \underbrace{(d \cdot \sigma + n)}_{\text{visit}})$ pointers

$O(d \cdot \sigma) \rightarrow d \geq n$

Prefix search on trie: $O(d_s)$

Hash table (hashing with chaining)



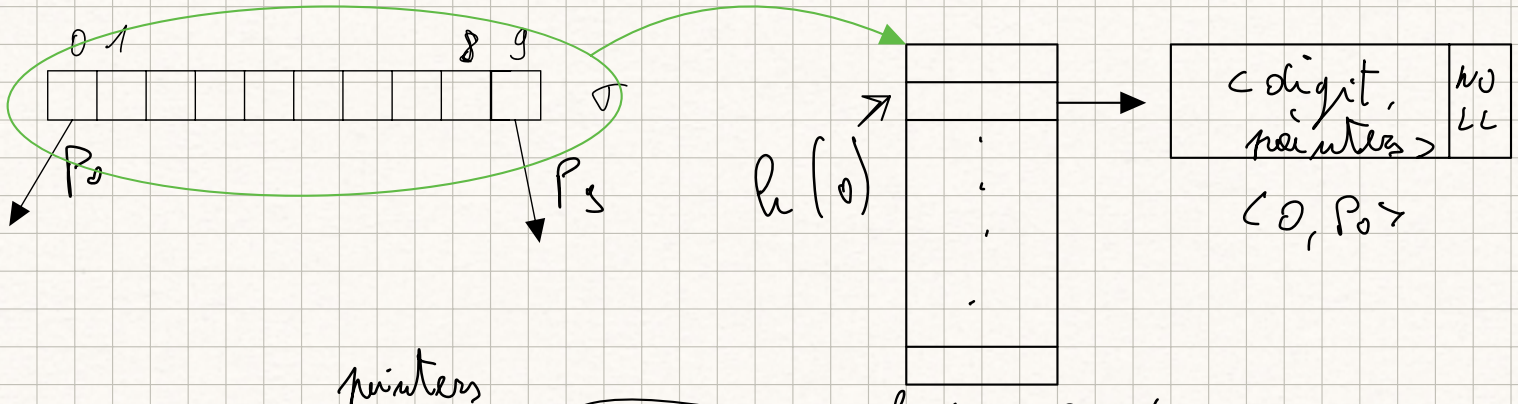
set $m = O(n)$ → # keys

$2 = O(1)$ ⇒ time for insertion $O(1)$

Store k keys taking $O(k)$ time, and space

expected

Hash table for Trie:



pointers

space $= O\left(n + \sum_{m} \ell_m\right) \leq n + d = O(d)$ — # of edges leaving n

size of all hash tables $\leq n + d$

$$\text{Time} = O\Big(\underbrace{n + d}_{\text{construction time}} + \sum_u \ell_u \cdot \log \ell_u\Big) \le O(d \log \sigma)$$

sorting edges (hash table doesn't sort items)

$(d+n)\log d > n$

$$\le \sum_u \ell_u \cdot \log \sigma = (\log \sigma) \cdot \sum_u \ell_u = d \log \sigma$$

Time to sort with an hash table + trie $= O((d+n)\cdot \log \sigma)$

LSD - first Radix Sort (use counting sort that is stable)

			1°			2°			3°		
0	1	7		1	1	1		0	0	7		0	0	7
0	4	2		0	4	2		1	1	1		0	1	7
6	6	5	→	6	6	5	→	0	1	7	→	0	4	2
1	1	1		0	1	7		0	4	2		1	1	1
0	0	7		0	0	7		6	6	5		6	6	5

MSD $\begin{cases} O(d \log \sigma) \\ O(d) \text{ space} \end{cases}$

- L phases
- $O(n+\sigma)$ time per phase

$\downarrow$

$O(L\cdot(n+\sigma))$ time

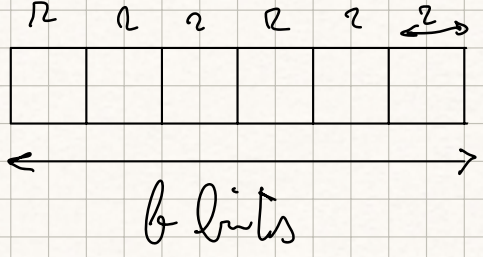
$\downarrow$

$O(L\cdot n + n\cdot\sigma) = O(N)$ time

N = tot string len.

Assume that all keys are binary

- every phase cost: $O(n + 2^r)$ time
- $\frac{b}{r}$ groups



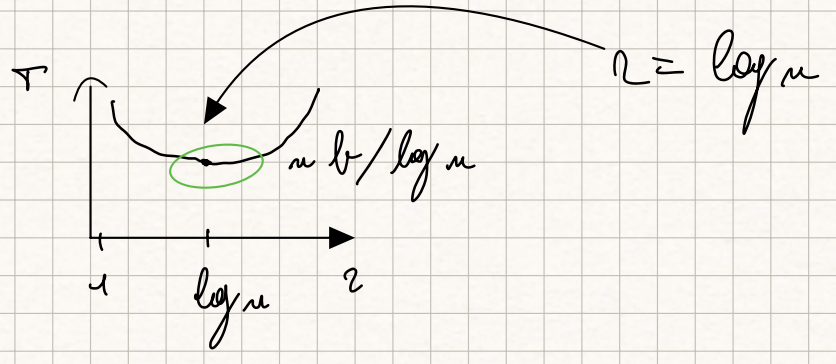
- # phases $= \frac{b}{r}$ = # of letters in a word

2^r = alphabet size

$$O\left(\frac{b}{r} \cdot (n + 2^r)\right) \text{ Time}$$

if: $r \leq \log_2 n \longrightarrow 2^r \leq n$ Time compl. $= O\left(\frac{b}{r} \cdot n\right)$

if: $r > \log_2 n \longrightarrow 2^r > n$ Time compl. $= O\left(\frac{b \cdot 2^r}{r}\right)$



Min time compl. $r = \log_2 n \longrightarrow O\left(\frac{bn}{\log_2 n}\right)$ Time $= O\left(\frac{N}{\log_2 n}\right)$ Time ■

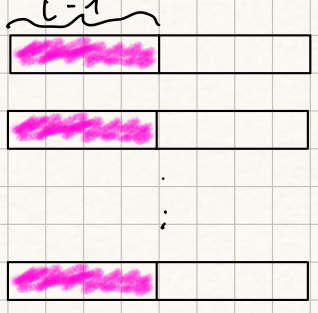
$\frac{N}{\log_2 n} \sim$ # of keys

LSD MSD

Multi-Key Quicksort: cmp-based sorter for strings: $\Omega(n \log n + d)$

```
MKQ(R, i)   i ≥ 1 ; R a set of strings;
    if |R| ≤ 1 return R;
    else:
        choose a pivot string in R → P
        R< = { S ∈ R : S[i] < P[i] }
        R= = { S ∈ R : S[i] == P[i] }
        R> = { S ∈ R : S[i] > P[i] }
        A = MKQ(R<, i);
        B = MKQ(R=, i+1);
        C = MKQ(R>, i);
        return A+B+C
```

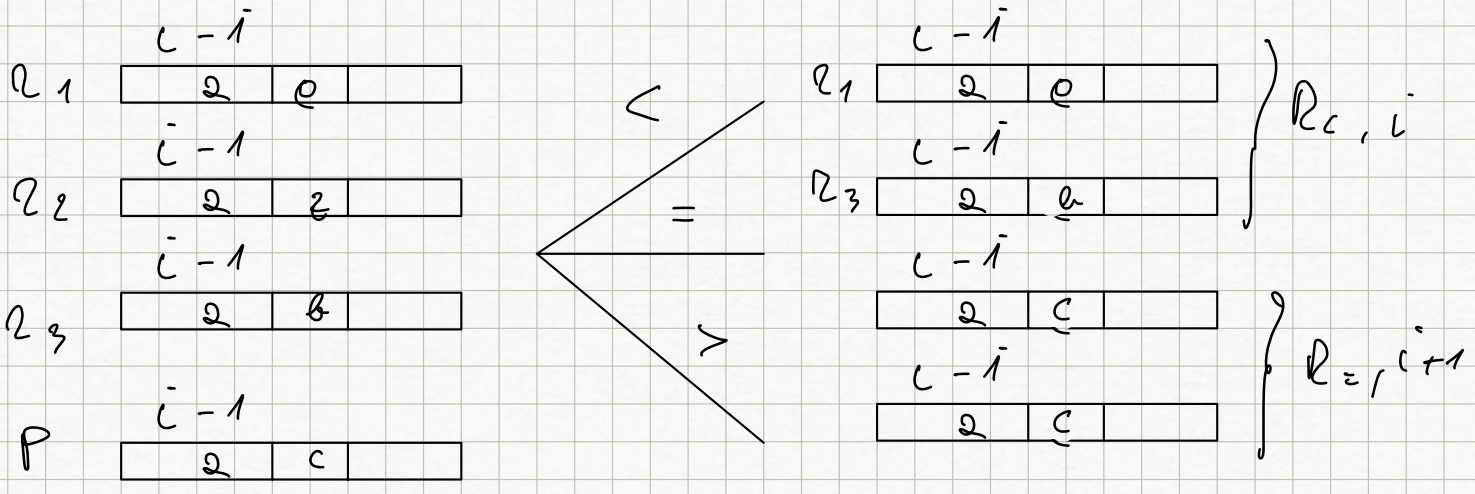
Invariant:



are equal

prefix free = no string in R prefixes any other string in R

. Assume that invariant holds



. Assume that invariant holds (R, i)

Let consider a string $S \in R$

$S \in R \rightarrow R_<, R_> \rightarrow |R_<| < |R|, |R_>| < |R|$

$S \in R \rightarrow |R_=| \leq |R|, i \rightarrow i+1$; $i \leq |S|$ string len. ; $i \leq d_S$

in the expected case, the pivot splits R in a way that $|R_<| \cup |R_>| \simeq \frac{|R|}{2}$

· $S \in R_<$ or $R_>$, $O(\log n)$ times comparison

· cost of processing S in $O(\log n + d_S)$ time

$\leq \sum_S O(\log n + d_S) = O(n \log n + d)$ lower bound

S = (cat, abi, cart, car, at)

i = 1 P = cart

R = S

R< ABI, AT } R<, 1

P = ABI

R< = ∅

R= = ABI, AT } R=, 2

R> = ∅

P = ABI

R< = ∅

R= = ABI } R=, 3

R> = AT } R>, 2

R= CAt, CAST, CAR } R=, 2

P = CAT

R= CAT, CAST, CAR } R=, 3

P = CAT

R< CAST, CAR } R<, 3

R= = CAT R=, 4

R> ∅

P = CAST

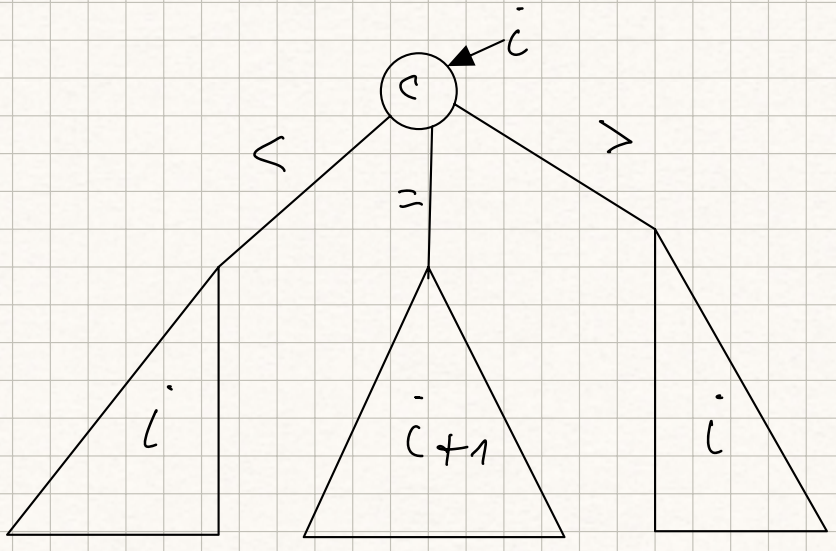
R< CAR

R= CAST

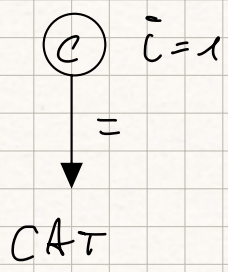
R> ∅

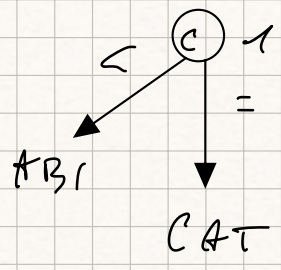
R> = ∅

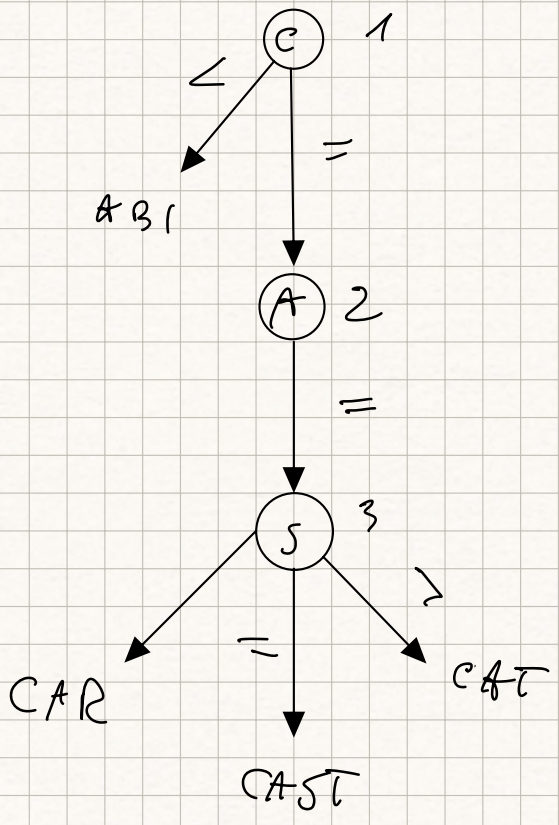
Ternary Search Tree (TST)

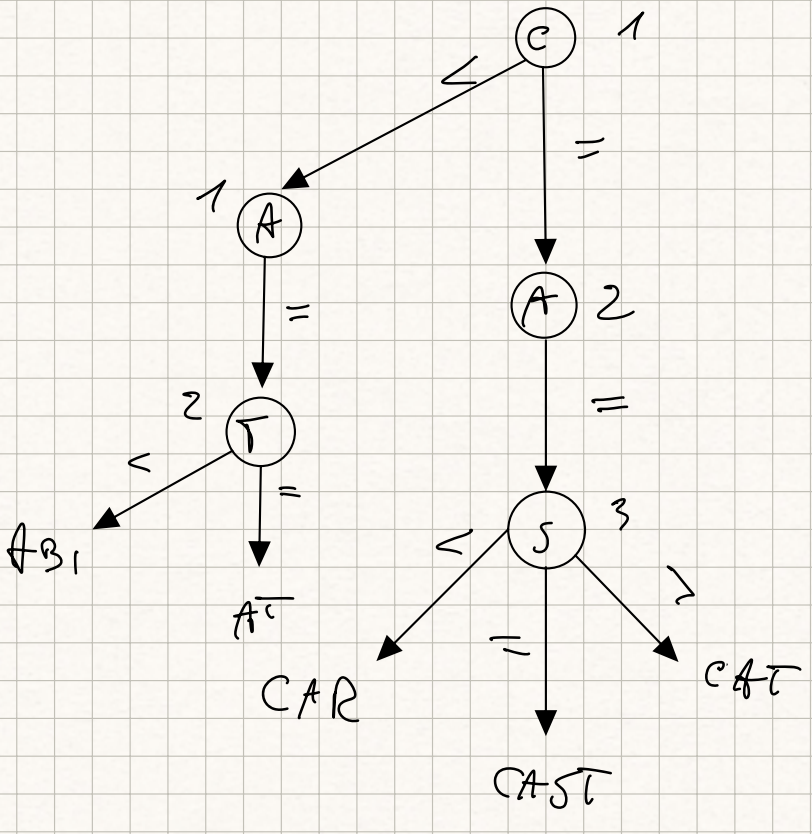


S = (cat, abi, cart, car, at)









If TST is balanced → search P : $O(|P| + \log n)$

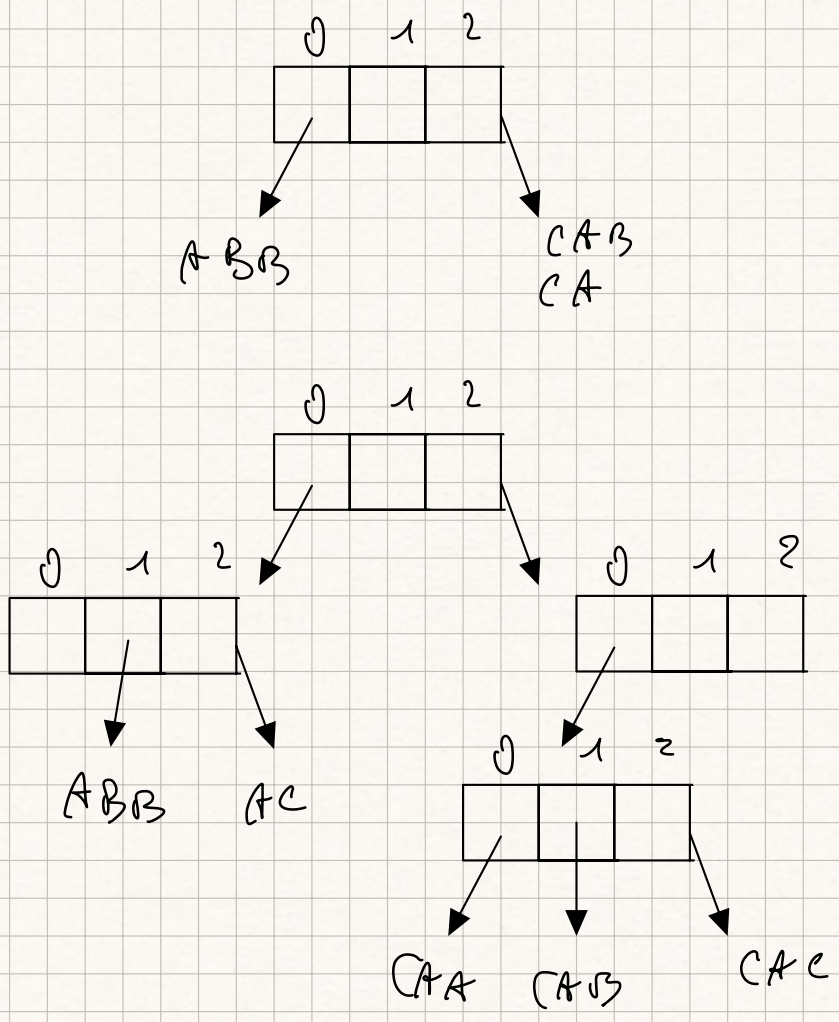
length of searched string

base 2 bound of comparison tree

build a tree = make multi-key qsort = $O(N + n \log n)$

$S = (ca\,b,\ abb,\ cca,\ cac,\ ac)$ $\quad \Sigma = \{a, b, c\}$ alphabet $\{0, 1, 2\}$

trie, where fan-out is implemented by an array



$S = \{ABA, ABBC, ABBD, BA\}$

- uncompacted trie
- TST
- TKR [pivot is always the first string of R]

Interpolation Search

	1	2	3	4	5	6	7	8	9	10	11	12
$X =$	1	2	3	8	9	17	19	20	28	30	32	36
	B_1			B_3		B_6		B_8	B_{10}		B_{11}	B_{12}

$n = \#\text{keys} \in \mathbb{N} \Rightarrow x_1 < x_2 < \dots < x_n$

build once use many times

Query $(y) \rightarrow y \in X$? $\longrightarrow$ binary search $O(\log n)$ time

Idea: • Partition X in a "logical way" into buckets

$B_1, B_2, \dots, B_n$

• Size of each bucket is $b = \dfrac{x_n - x_1 + 1}{n} = \dfrac{36 - 1 + 1}{12} = 3$

(x_n: biggest elem., x_1: smallest elem., universe: dim of the universe)

Portions of X

new array $I =$

1	3	0	0	4	5	
B_1		B_2		B_3		

$|I| = 2n$

build once use many times

# of bucket = # of items

Search (y): $j = \left\lfloor \dfrac{y - x_1}{b} \right\rfloor + 1$ $O(1)$

binary search in $B_j = X[I[2j-1], I[2j]]$

worst case $O(\log_2 \min\{n, b\})$

On expect. $O(\log_2 \text{const}) = O(1)$

Th x obvious at random

$\mathbb{E}$ [max # keys per bucket, when we have n buckets] $= O(\log^2 n)$

$O(\log_2(\log^2 n))$ time w.h.p $= O(\log(\log(n)))$ time

Time complexity is $O(\log_2 \Delta)$, where $\Delta = \frac{\max_{i=2,\dots,n} x_i - x_{i-1}}{\min_{i=2,3,\dots,n} x_i - x_{i-1}}$

measure of the distribution of the keys $x_1, x_2, \dots, x_n$

for Interpolation search

Dim:

① $\max_{i=2,\dots,n} x_i - x_{i-1} \geq \frac{\sum_{i=2}^{n} x_i - x_{i-1}}{n-1} \geq \frac{1 + \sum_{i=2}^{n} x_i - x_{i-1}}{n} =$

media

Telescopic sum

$x_n - x_{n-1} + x_{n-1} \dots \dots x_1$

$= \frac{1 + x_n - x_1}{n} \leq b$

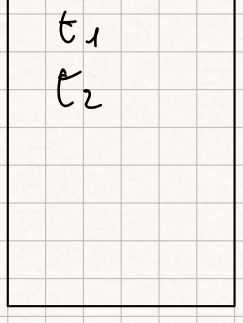
② $|B_S| \leq \frac{b}{\min_{i=2\dots n} x_i - x_{i-1}} \overset{①}{\leq} \frac{\max x_i - x_{i-1}}{\min x_i - x_{i-1}} = \Delta$

Set intersection $A \cap B$

A_n and B_m are sorted

$m \leq n$

Dict



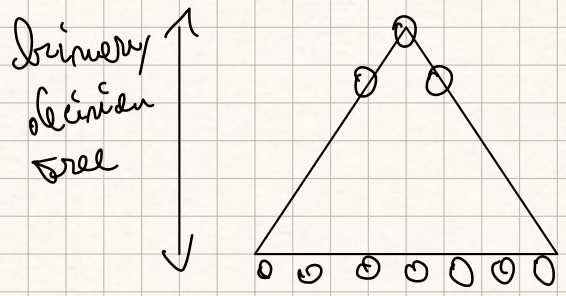
→ 1, 3, (5), 11

→ 2, 4, (5), 10, 12

- Merge based: $O(n+m)$ times optimal when $n \approx m$
- Binary search: $O(m \log_2 n)$ times i search the shorter in the longer list with B. S.

if: $m \log n < n \rightarrow m < \frac{n}{\log n} \rightarrow$ Binary search optimal

else: Merge search



A . . . . . n

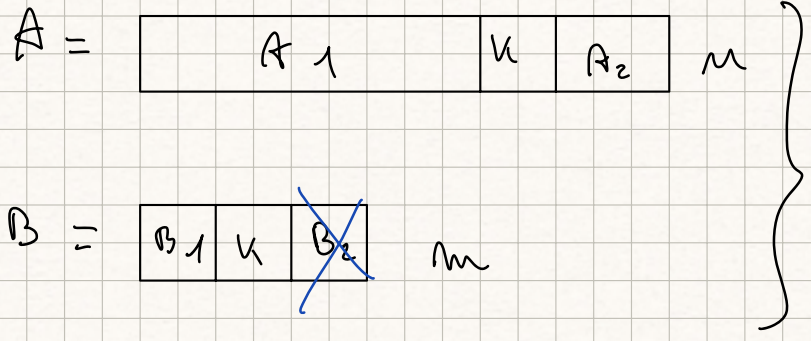
B . . . m

$\binom{n}{m}$ ways that B can be included in A

#nl →

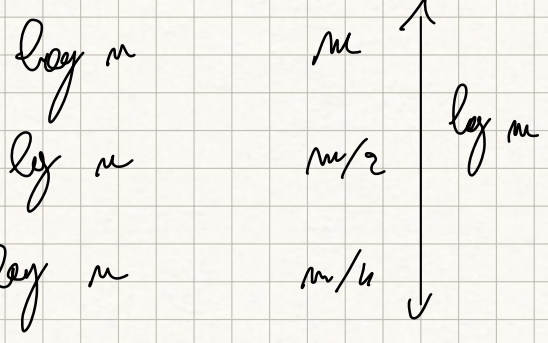
$$\binom{n}{m} \#\text{comp} \geq \log_2 \#nl \geq \log_2 \binom{n}{m} \geq \Omega\left(m \log_2 \frac{n}{m}\right)$$

- Mutual partitioning



① $(\log n)(\log m)$ time

unbalanced partition (last case)



② $T(n,m) = O(\log(n)) + 2T(n/2, m/2) = O(m(1+\log\frac{n}{m}))$

balanced partition $\quad m \le n$

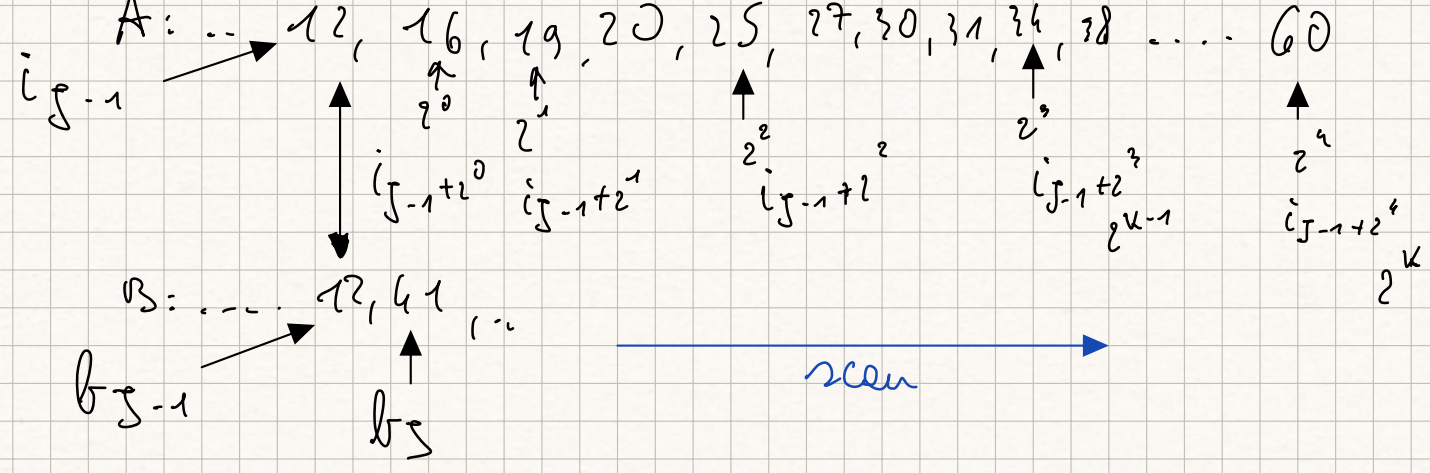
- $m \approx n \quad O(m) = O(n) \longrightarrow$ as merge based
- $m \ll n \quad O(m \log n) \longrightarrow$ as binary search based

LOWER BOUND

$$T(n,m) = \Omega(\log n) + 2T\left(\frac{n}{2}, \frac{m}{2}\right) = \Omega\left(m \log \frac{n}{m}\right)$$

Doubling search, exponential search, ...

$\Delta_j = \min(n, 2^{k-1}) \le 2^{k-1}$



① $A[i_{j-1} + 2^{k-1}] < b_j \le A[i_{j-1} + 2^k]$

② $2^{k-1} < i_j - i_{j-1} \le 2^k \qquad \min(i_{j-1}+2^k, m) > i_j \ge i_{j-1} + 2^{k-1}$

③ $\Delta_j \le 2^{k-1} < i_j - i_{j-1} \le 2^k$

④ $\log_2 2^{k-1} \leq \log_2 (i_j - i_{j-1})$

$\Downarrow$

$k-1 \leq \log_2 (i_j - i_{j-1})$

$\Downarrow$

$k = O(\log_2 (i_j - i_{j-1}))$

$\leq k + k = O(k)$ steps we perform

Block $O(\log_2 (i_j - i_{j-1}))$

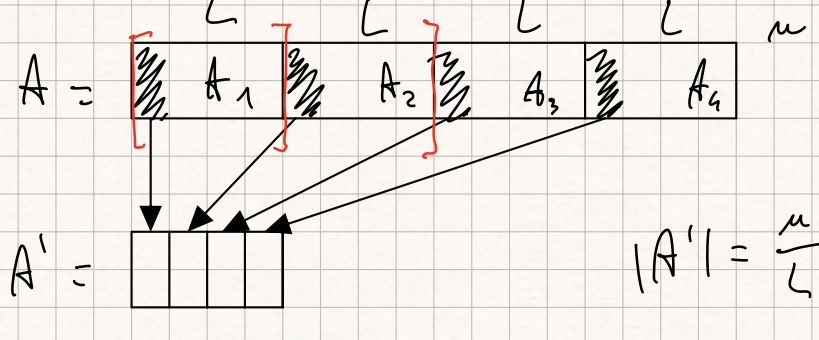
Time $= \sum_{j=1}^{m} O(\log_2 (i_j - i_{j-1})) = O\left(\sum_{j=1}^{m} \log_2 (i_j - i_{j-1})\right)$

↑ Jensen Inequality

Telescopic sum: $i_m - i_0 \leq n$

$= O\left(\sum_{j=1}^{m} \log_2 (i_j - i_{j-1})\right) = O\left(m \cdot \log_2 \frac{n}{m}\right)$

Two level storage approach



1) $A'[i+1] = A[1+iL] \qquad \forall i = 1, \dots, L$

2) merge $(A', B) \Rightarrow B_i = \{b \in B : A'[i] \le b < A'[i+1]\}$

·) $\forall i \quad A_i \cap B_i \qquad i = 1, 2, \dots, \frac{n}{L}$

$T_1 = O(|A'| + |B|) = O\left(\frac{n}{L} + m\right)$ time — Merge sequences

$$T_2 = \sum_{i=1}^{n/L} (|A_i| + |B_i|) = \underbrace{\sum_{i=1}^{n/L} |A_i|}_{\text{if } A_i \text{ picked} \Rightarrow n \cdot L} + \underbrace{\sum_{i=1}^{n/L} |B_i|}_{m} \le O(n + m \cdot L)$$

$O(T_1 + T_2) = O\left(\frac{n}{L} + m \cdot L\right)$ time

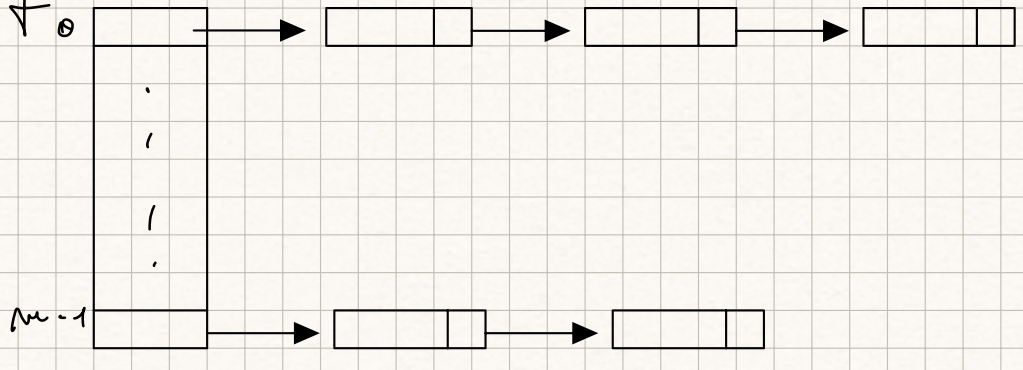
Hashing => Dictionary problem

$$D = \{o_1, o_2, \dots, o_m\}$$

o_i → key, satellite data

- membership(k) / search(k)
- insert(o)
- delete(k)

hashing with chaining



$h : U \to [m]$

collision resolution = chaining

Hashing useful iff $|U|$ is large

.) h is simple uniform 'hash function'

. every key can go in every slot with uniform prob.

$$h(k) \in \{0, 1, \dots, m-1\} \qquad P(h(k) = i) \to 1/m$$

$$E[\text{list } T[i]] = \sum_{k \in D} 1 \cdot \underbrace{P(h(k) = i)}_{1/m} = \sum_{k \in D} \frac{1}{m}$$

load factor $= n/m = \alpha$

$n = \#$ keys

$m = \#$ hash entries

• search compl. $O(1) \iff \alpha = O(1) \iff n \simeq m \to n = c \cdot m$

search $(x) = O(1 + \alpha)$

$n \geq m \longrightarrow \alpha \leq 1$

$n < m \longrightarrow \alpha > 1$

• Global rebuilding technique

n insertions	20	21	22	...	40
m	20				20
2	1				2
# ops.					

when n is double of m i stop and i build $T' = 2 \cdot m$ and i restart all the operat.

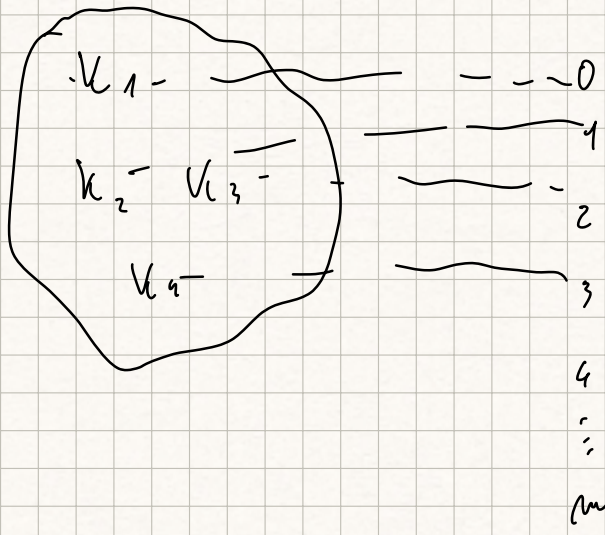
Amortized time complexity

. uniform single hashing

$h_i : U \longrightarrow [m]$

$\forall k \xrightarrow{h_i}$ any value in $[m]$

$\#h_i = \#$ all possible mappings from U to $[m]$



key	k_1	k_2	k_3	...	k_U
# ways to map	m	m	m	...	m

mapping in total is $\underbrace{m \cdot m \cdot \ldots \cdot m}_{|U|} = m^{|U|} \rightarrow \log_2 m^{|U|}$ bits

$= |U| \cdot \log_2 m$ bits

2^{64}

requires too much space

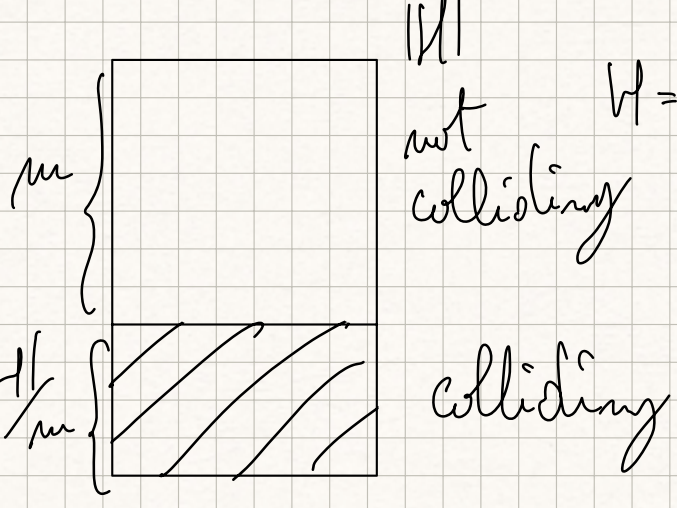
Universal hashing

. pairwise independence

. the class of hash function is called universal

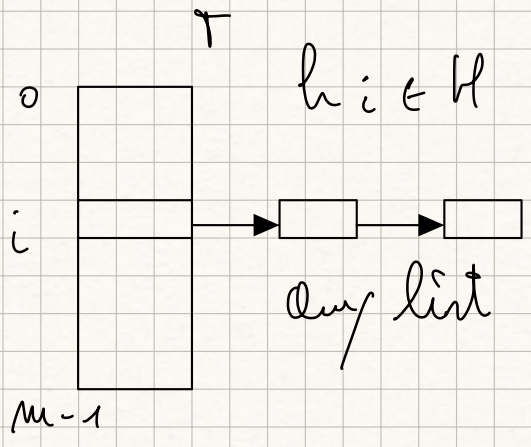
iff $\forall x, y \in U, x \neq y \quad |\{h \in H : h(x) = h(y)\}| \leq \frac{|H|}{m}$

$P[h \text{ collides for } x, y] = \frac{|H|/m}{|H|} \leq \frac{1}{m}$



$H = \{h : U \to [m]\}$

Class H exists → design hashing with chaining



Th: under the previous assumption, there exist one hash table with chaining of size m, in which search(k) over n objects takes $O(1 + n/m)$ expected time.

Dim:

$E[\text{length } T[i]] = \sum_{k \in D} 1 \cdot P[h(k) = i] = 1 + \sum_{k \in D} 1 \cdot P_h[h(k) = h(k')] =$

$E[\text{length } T[i]]$: expected length of $T[i]$

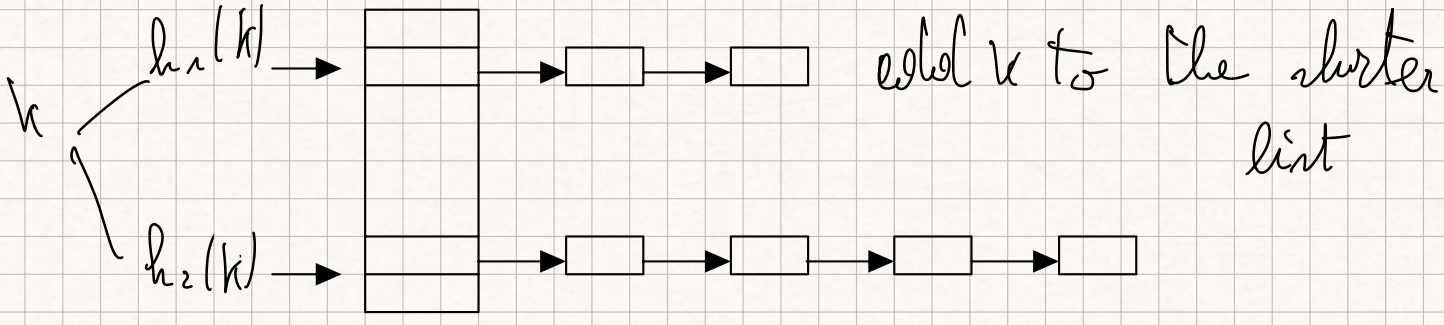
$k' \neq k;\ h(k') = i$

$$= 1 + \sum_{\substack{k \in D \\ k \neq k'}} \frac{1}{m} = 1 + \frac{n-1}{m} < 1 + \frac{n}{m} = 1 + \alpha$$ ■

$\frac{1}{m}$ → def of universal class

- expected len of a list in T is $O(\alpha)$

$h_1, h_2 \in H$



2 hashes → $O\left(\frac{\log n}{\log \log n}\right)$ length of the longest list in T × 1 hash function

d hashes → $O\left(\frac{\log \log n}{\log_2 d}\right)$ length of the longest list in T × d hash function

$d \geq 2$

$$H = \{ h_a(k) = \sum_{i=0}^{r-1} a_i \cdot k_i \bmod m, \text{ where } a \in (0, |U|-1] \}$$
$a \in \mathbb{N}$

Universal hash function

Prime

k | k_0 | k_1 | ... | k_{r-1} | $\log_2 |U|$ bits

$\leftrightarrow$ $\log_2 m$ bits

$r = \frac{\log_2 |U|}{\log_2 m}$ # of partitions for k, a

$k_i \in \{0, 1, \dots, m-1\}$

a | a_0 | a_1 | | a_{r-1}

$|H| = |U| - 1$

One function per value of a

Prove that H is universal: means that, given $x \neq y$

$$\#\, h_a : h_a(x) = h_a(y) \leq \frac{|H|}{m} = \frac{|U|-1}{m}$$

Pick $x \neq y$

x = | x_0 | | x_{r-1}

$\neq$

at least $x_0 \neq y_0$

y = | y_0 | | y_{r-1}

if $h_a(x) = h_a(y)$ then $\sum_{i=0}^{r-1} a_i x_i \bmod m = \sum_{i=0}^{r-1} a_i y_i \bmod m$

$$\sum_{i=0}^{r-1} a_i (x_i - y_i) \equiv 0 \bmod m \rightarrow a_0 (x_0 - y_0) + \sum_{i=1}^{r-1} a_i (x_i - y_i) \equiv 0 \bmod m$$

$$a_0(x_0 - y_0) \equiv -\sum_{i=1}^{r-1} a_i(x_i - y_i) \pmod{m}$$

$$a_0 \equiv -(x_0 - y_0)^{-1} \cdot \sum_{i=1}^{r-1} a_i(x_i - y_i) \pmod{m}$$

a_0 ← $a_1, a_2 \dots a_{r-1}$

$\#a$

$\#\, a : h_a(x) = h_a(y)$

dim of $|U|$ for $a = m^2$

exclude a_0

$r-1$

$\#\, a$ that give a collision : $m^{\underbrace{r-1}_{a \neq 0}} = \frac{m^2}{m} - 1 =$

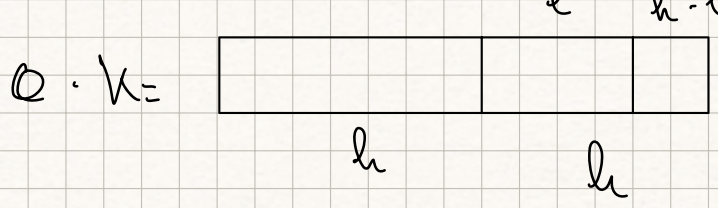
$$= \frac{|U|}{m} - 1 = \frac{|U| - m}{m} < \frac{|U| - 1}{m} \quad \blacksquare$$

$h_{a,b}(x) = (ax + b) \bmod m$ ← prime $= ((ax + b) \bmod m) \bmod 2^l$ $m > U$

$h_a(x) = (\underbrace{a \cdot x}_{2 \cdot h \text{ bits}} \bmod \underbrace{2^h}_{U}) \operatorname{div} 2^{h-l}$

$h = 64$ 20 6

$m = 2^{l-20}$



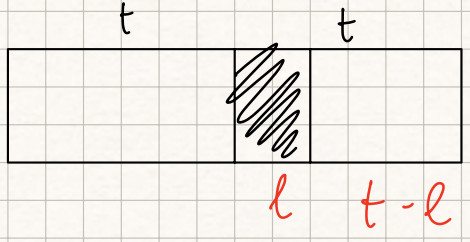
scarto il primo blocco con $\bmod 2^h$

scarto l'ultimo blocco con $\operatorname{div} 2^{h-l}$

$h(x) = (a \cdot x \bmod 2^t) \operatorname{div} 2^{t-\ell}$

$a \in U$; $|U|$; integer in $[2^\ell]$ = m

$h : U \longrightarrow [m]$



Cuckoo hashing:

Query $O(1)$ time

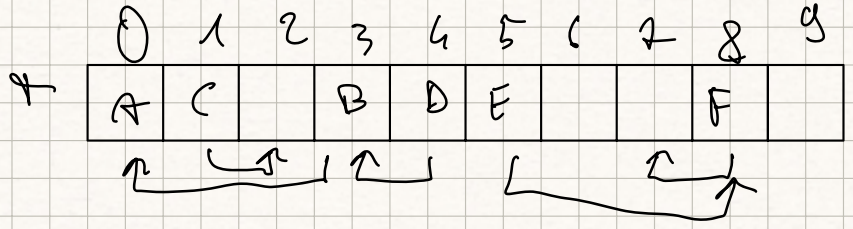
deletion $O(1)$ time

insertion $O(1)$ time amortized expected

Two choices : h_1, h_2

graph "logical"

keys can move : relocated



let's insert G $h_1(G)$ $h_2(G)$

= 0 = 3

Th: $\forall i, j$ positions in $[m]$, $\forall c > 1$, if $m \geq 2cn$ ($|T|$ = dim hash table; n = #keys)

then $P\left(\exists \text{ undirected shortest path } i \text{ — } j\right) \leq \frac{1}{m \cdot c^L}$

exp small in L

$\alpha = \frac{n}{m} \leq \frac{1}{2c} < \frac{1}{2}$

n = # edges = # keys

m = # nodes = table size

L = length of the shortest path ≥ 1

Proof: by induction on L

$L = 1$: $P(i \text{ — } j) = \frac{1 \text{ edge}}{m}$

$$P\left(\exists k : \begin{matrix} h_1(k) = i, & h_2(k) = j \\ h_1(k) = j, & h_2(k) = i \end{matrix}\right) \leq \sum_{k \in D} P\left(\begin{matrix} h_1(k) = i, & h_2(k) = j \\ h_1(k) = j, & h_2(k) = i \end{matrix}\right) \quad //$$

$h_{1,2}$ = class of universal hash function

$$\sum_{k \in D} 2 \cdot \frac{1}{m} \cdot \frac{1}{m} \leq 2 \sum_{k \in D} \frac{1}{m^2} = \frac{2n}{m^2} \leq \frac{n}{2c} \cdot \frac{2}{m^2} = \frac{1}{mc} = \frac{1}{m c^1}$$

($n \leq \frac{m}{2c}$)

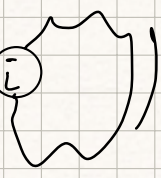
$L > 1$: $P(i \overset{L}{\text{—}} j) = P(\exists z : i \overset{\text{S.P.}}{\underset{L-1}{\text{—}} z \underset{1}{\text{—}}} j) \leq \sum_{\text{nodes } z \ (|T| = m)} P(i \overset{L-1}{\text{—}} z \overset{1}{\text{—}} j)$ $\quad //$ $\quad //$

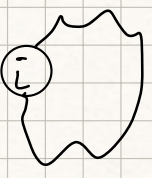
$\frac{1}{m \cdot c^{L-1}}$ $\quad$ $\frac{1}{mc}$

$$= \sum_z \frac{1}{m \cdot c^{L-1}} \cdot \frac{1}{mc} = \sum_z \frac{1}{m^2 \cdot c^L} = \frac{m}{m^2 \cdot c^L} = \frac{1}{m \cdot c^L}$$

■

$n = \#\text{keys} \qquad |T| = m$

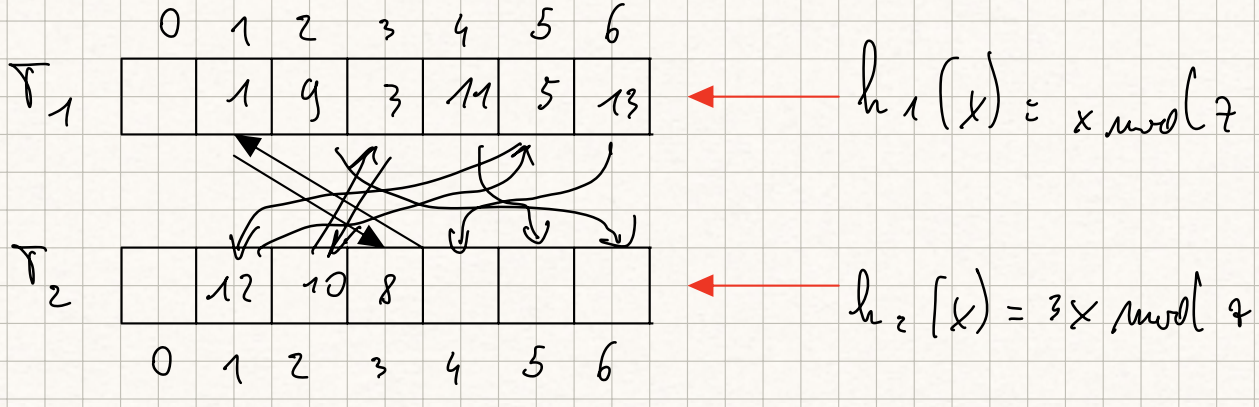
$P(\text{fail insertion}) = P(\text{2 cycles}) \leq P(\exists \text{ cycle}) = P(\exists i : $  $)$

$$P(\exists i, \exists L : \text{  }) \leq \sum_{\substack{\text{nodes} \\ i=1}}^{m} \sum_{L \geq 1} \frac{1}{m \cdot c^L} = \sum_{L \geq 1} \frac{m}{m \cdot c^L} = \sum_{L \geq 1} \left(\frac{1}{c}\right)^L = \frac{1}{c-1}$$

$c = 3 \qquad m \geq 2cn$

Corollary: Given n keys and table of size m, if we take $m \geq 6n$ then the $P(\text{fail}) \leq 1/2$

↳ $\alpha = \frac{n}{m} \leq 1/6 \approx 15\%$



$S = (1, 5, 8, 3, 17, 10, 11, 13, 9)$

1 3 5 1 3 2 6 4 2 6

first: fill T_1, then T_2 if $T_1[i]$ is full

Minimal Ordered perfect hashing

$m \geq n$

no collision

$\forall k', k'' \in D : h(k') \neq h(k'')$

$\forall k' < k'' \in D : h(k') < h(k'')$

rank(k')

$m = 9$

Key	h (goal)	h_1	h_2
chrome	0	1	6
cat	1	7	2
day	2	5	7
flop	3	4	6
home	4	1	10
house	5	0	1
ran	6	8	11
trip	7	11	9
tan	8	5	3

① $h_1, h_2 : U \longrightarrow [m']$ prime number

$m' > m = n$ (universal)

$m' = 13$, $m = 9$

$a \cdot k + b \mod m'$

randomly chosen

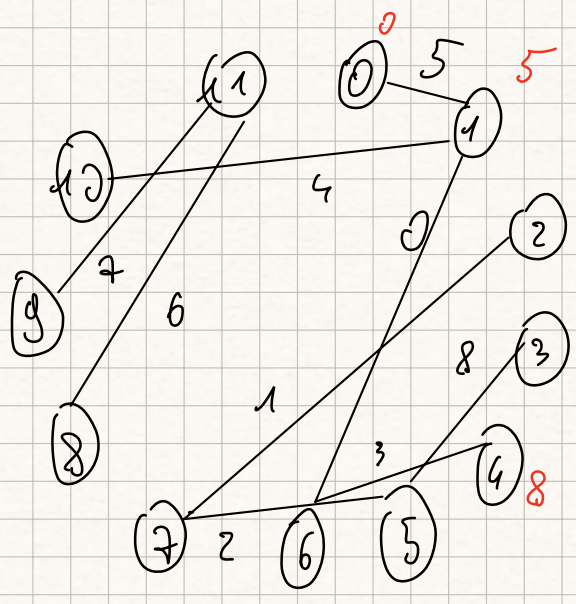
② $g(k) = [m'] \longrightarrow [n]$ hash to be computed

we want to compute g so that

$h(x) = [g(h_1(x)) + g(h_2(x))] \bmod n$

We build G and check if it is acyclic → ok

NO → restart from scratch



To define g it is:

$h(\text{abacus}) = [g(h_1(\text{abacus})) + g(h_2(\text{abacus}))] \bmod q$

① $0 = g(1) + g(6) \bmod q$

② $1 = g(7) + g(2) \bmod q$

$5 = g(0) + g(1) \bmod q$

We always start with the biggest connected components

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

assign the first value to a given vertex or we prefer (typically

$y =$

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

space $= m' \cdot \log m + O(n)$ bits

Ex: $D = \{AA, BB, BD, CD\}$

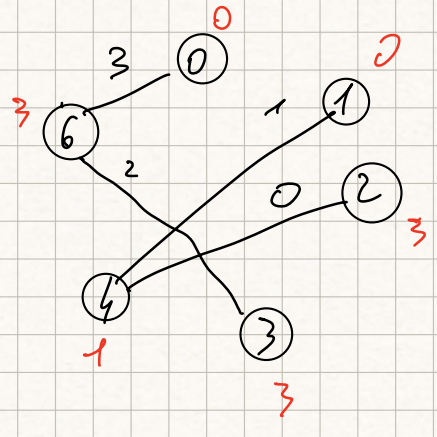
	A	B	C	D
2	1	2	3	4

$h_1(xy) = 3 \cdot 2(x) + 2(y) \bmod 7$

$h_2(xy) = 2(x) + 2(y) \bmod 7$

m' should be greater than # keys $7 > 4$

	h	h_1	h_2
AA	0	4	2
BB	1	1	4
BD	2	3	6
CD	3	6	0



$g =$

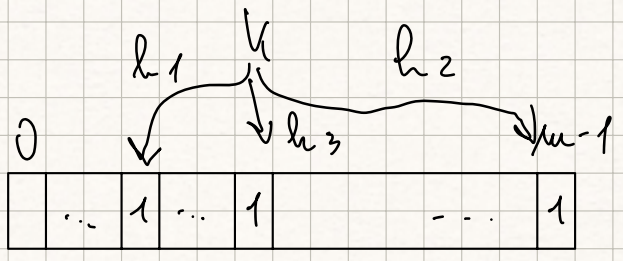
0	0	3	3	1	0	3
0	1	2	3	4	5	6

$h(AD) = 0$

$$\underbrace{g(\underbrace{h_1(AD)}_{0})}_{0} + \underbrace{g(\underbrace{h_2(AD)}_{5})}_{0} \bmod 4$$

Bloom Filter (dictionary)

- membership
- insertion
- ~~deletion~~



① binary array $m > n$

② $h_1, h_2, \ldots, h_2 : U \to [m]$

$$\text{membership}(k) = B[h_1(k)] \wedge B[h_2(k)] \wedge B[h_3(k)] \wedge \ldots \wedge B[h_2(k)]$$

1 yes ≈ maybe

0 NO for sure 100%

Keys are dropped after the insertion

$$P(B[i]=0, \text{after inserting } n \text{ keys with } 2 \text{ hash fuctions}) \approx \left(\frac{m-1}{m}\right)^{2 \cdot n} = \left(1 - \frac{1}{m}\right)^{2 \cdot n}$$

$$P(\text{error}) = P(y \notin D \text{ but } B[h_i(y)] = 1, \forall i = 1, \ldots 2) = \prod_{i=1}^{2} P(B[h_i(y)] = 1)$$

$$= \prod_{i=1}^{2} \left(1 - \left(\frac{m-1}{m}\right)^{2 \cdot n}\right) = \left[1 - \left(\frac{m-1}{m}\right)^{2 \cdot n}\right]^2 = \left[1 - \left(1 - \frac{1}{m}\right)^{m \frac{2 \cdot n}{m}}\right]^2 =$$

$\left(1 - \frac{1}{m}\right)^{m} \approx e^{-1}$

$$\approx \left[1 - e^{-\frac{2 \cdot n}{m}}\right]^2$$

■

Fixed n, m find c s.t. $P(\text{error})$ is minimum

$$\Rightarrow \min_{\hat{c}} \left(1 - e^{-\frac{2 \cdot n}{m}}\right)^{c}$$

$$\hat{c} = \frac{m}{n} \ln 2$$

$$P(\text{error with } \hat{c}) = \left(\frac{1}{2}\right)^{\hat{c}} = \left(\frac{1}{2}\right)^{\frac{m}{n} \ln 2} = (0{,}6185)^{m/n}$$

(1) Fix $m \rightarrow \hat{c}$, guarantees minimum error for that n, m

(2) Fix $\hat{c}$, $\hat{m} \rightarrow P(\text{error with } \hat{c}, \hat{m})$

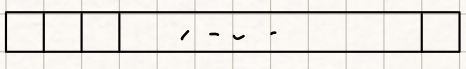
$\hat{c}$: speed, $\hat{m}$: memory

$\frac{m}{n} \approx$ # bits per key in BF

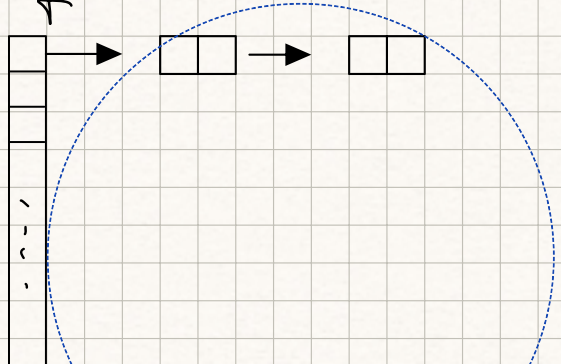
$m = 10$ M bits

$n = 1$ M keys

$\Big\}$ $\frac{m}{n} = 10$



T Cuckoo hashing $\geq m' \cdot \log_2 |U|$ bits



$\geq m' \cdot \log n + n(\log |U| + \log n)$

hashing with chaining: $m'' \cdot w + n(\log|U| + w)$

Let $m = c \cdot n$ where $c \geq 1$

using c bits per key as storage

- Cuckoo hash → $\frac{m'}{n} \cdot \log|U| > \log|U|$ bits per key
- Hashing with chaining, $m'' = n$ → $\frac{n \log n + n(\log|U| + \log n)}{n}$ bits per key

$= \log|U| + 2\log n > \log|U|$ bits per key

- BF takes c bits per key → $P(\text{error}) \sim (0{,}6)^c$

if $c \ll \log_2|U|$ BF is convenient

$m = c \cdot n$

$(0{,}6)^c$

$k = c \cdot \ln 2$

key size = key length

$|URL| \sim 1000$ chars $= 8000$ bits

Design a BF that minimizes P(error)

$$m = 2^{20} \qquad m = 2^{40}$$

$$\hat{n} = \frac{m}{n} \ln 2 = \frac{2^{40}}{2^{20}} \ln 2 = 2^{20} \ln 2$$

$$P(\text{error}) = (0,6185)^{2^{20}}$$

$\hat{n} = 10$ hash functions

$$P(\text{error}) = \left(1 - e^{-\frac{2m}{m}}\right)^2$$

Find m s.t. $P(\text{error}) \leq \frac{1}{1000}$

$$\left(1 - e^{-\frac{\hat{n}m}{m}}\right)^{\hat{n}} \leq \frac{1}{1000}$$

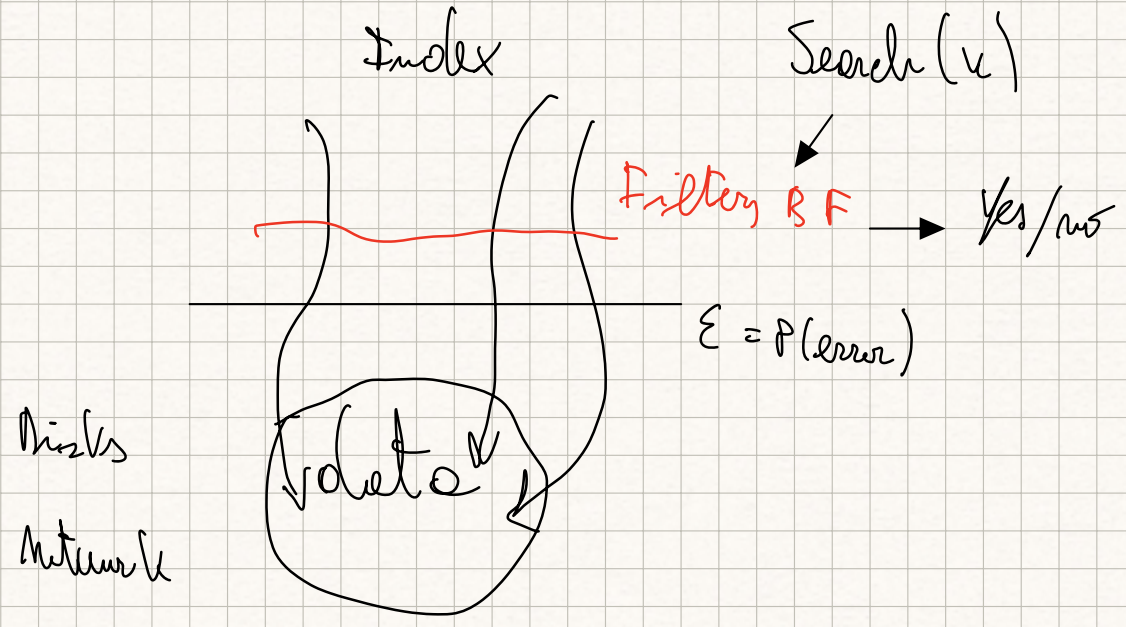
$$1 - e^{-\frac{\hat{n}m}{m}} \leq \sqrt[\hat{n}]{\frac{1}{1000}}$$

$$e^{-\frac{\hat{n}m}{m}} \geq 1 - \sqrt[\hat{n}]{\frac{1}{1000}}$$

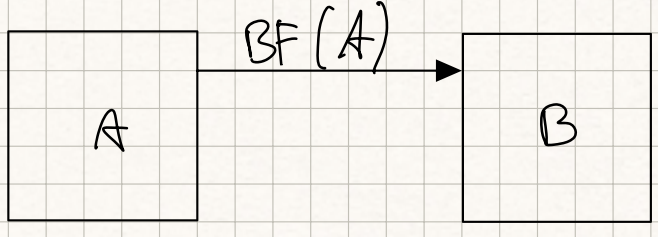
$$\frac{\hat{n}m}{m} \leq \ln\left(1 - \sqrt[\hat{n}]{\frac{1}{1000}}\right)$$

$$m \geq \frac{\hat{n}m}{\ln\left(1 - \sqrt[\hat{n}]{\frac{1}{1000}}\right)}$$

Data base (filters):



P2P (filters) : WAN (non volatile memory) $A \cap B$



- $|A| \log_2 |U|$ bits

$BF(A)$
- n_A bits
- 2_A hash function
- $n_A = |A|$

$\forall k \in B \rightarrow$ check if $k \in BF(A)$
- NO
- YES (MAYBE)

$MAYBE: V_i \in B \longrightarrow k \in A$

$\searrow \; k \notin A$

$(A \cap B) \cup (B - A)$

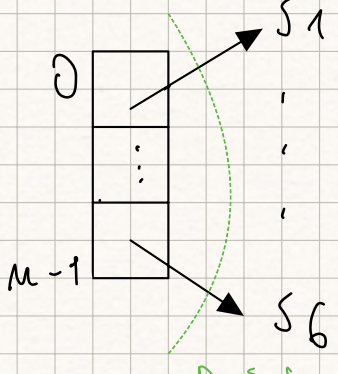
$$E[|B-A|] \leq P(\text{error}) \cdot |B| = (0{,}6185)^{\frac{m}{n}A} \cdot |B|$$

Prefix (string) search: given a dictionary of n strings of total length N build a data structure so that searching $P[1,p]$ as a prefix of the dictionary strings is fast

$D = \{\overbrace{000110}^{s_1}, \overbrace{0001110}^{s_2}, \overbrace{0010}^{s_3}, \overbrace{0011}^{s_4}, \overbrace{101}^{s_5}, \overbrace{111}^{s_6}\}$

$P = 0001 \longrightarrow$ 2 lexicographic searches $\rightarrow P\$$ ($\$$ = smaller than any symbol)

$\rightarrow P\#$ ($\#$ = bigger than any symbol)



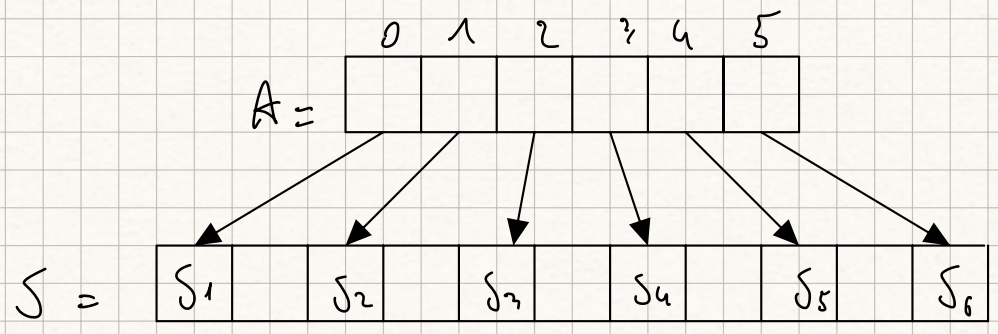
exact binary search $= O(p \cdot \log n)$ time

$O\left(\frac{p}{B} \cdot \log n\right)$ I/Os

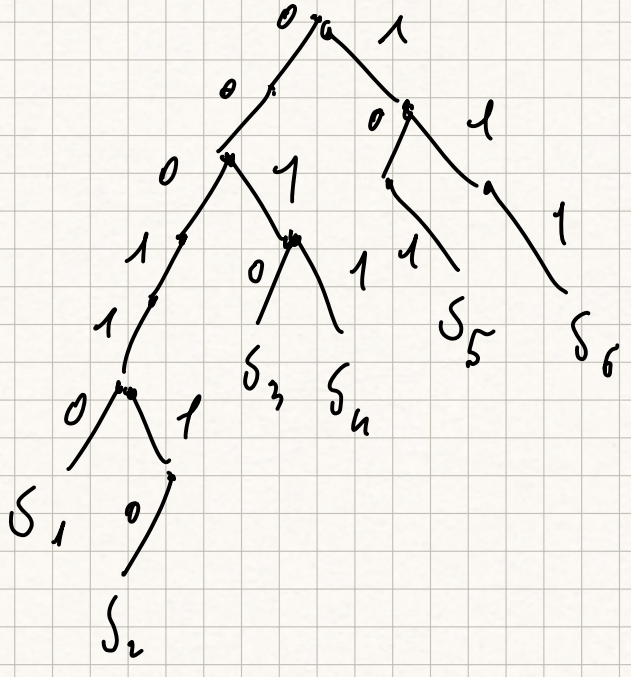
Space $= N + \left(1 + \frac{w}{\#\text{pointers}}\right) n$ bytes

$\uparrow$ $\uparrow$

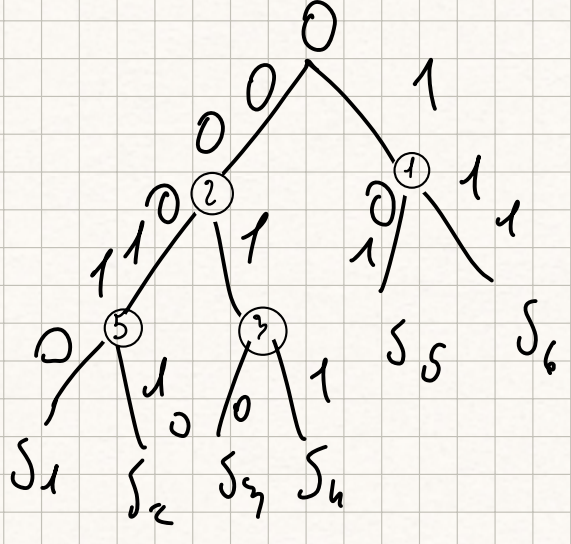
constant per string $\#$ strings



Uncompacted trie



Compacted trie:



n = leaves

internal nodes $< n$

edges $< 2n$

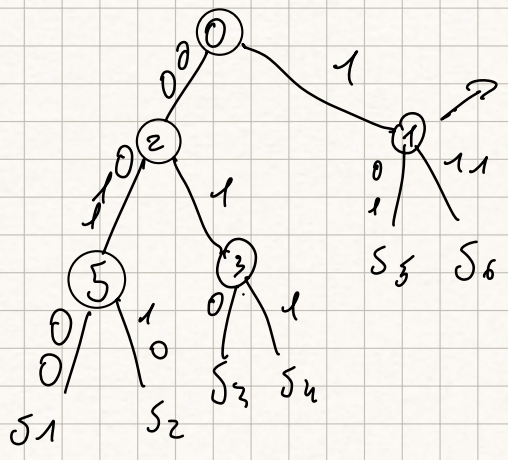
$O(n)$ space

Compacted trie saved as: < string pointer, start, end >

$\langle S_1, 1, 2 \rangle$; $\langle S_2, 3, 5 \rangle$ ...

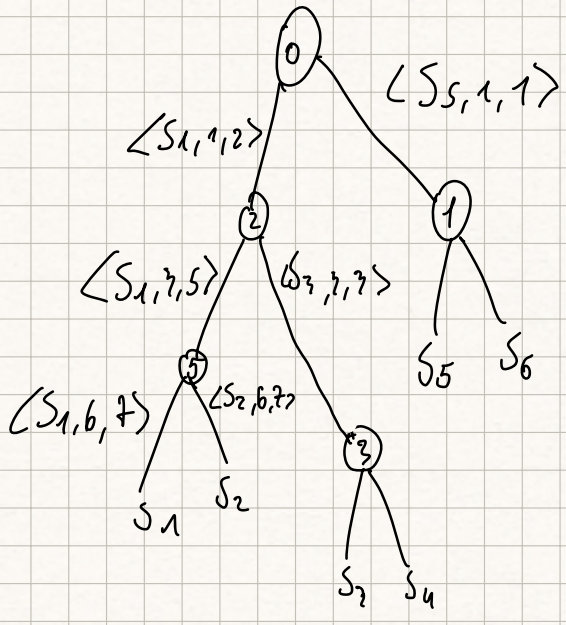
Patricia Trie:

$S = \{0001100, 0001110, 0010, 0011, 101, 111\}$

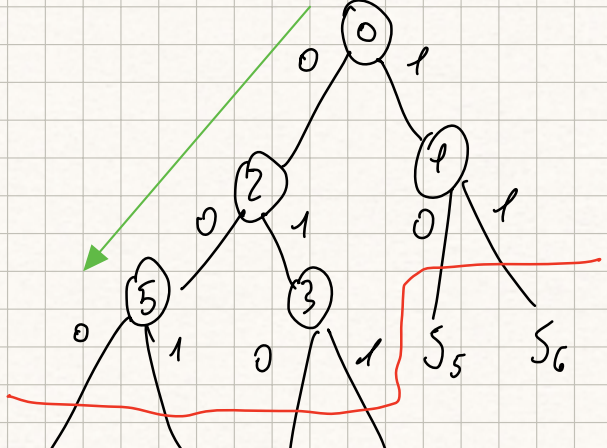


length of the string we look for

In order to fit in internal memory we save pointers:



Patricia trie:



Keep only first char of every edge

leaf

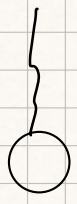
S_1 S_2 S_3 S_4

lexicographic search: $P = \underset{1}{\overset{0}{0}}\,1\,\underset{1}{\overset{2}{0}}\,0\,0\,\underset{1}{\overset{5}{0}}$

① Downward search: match P's chars as much as possible

P = ... B ... stop, pick a descending leaf

prefix



P is short: pick a descending leaf

② Compute the longest common prefix between P and the picked leaf. Execute $O(\frac{P}{B})$ I/Os

Execute $O(P)$ time

③ upward traversal

S =

B_1	B_2	B_3	B_4
AABAA	ABABAAA	ABABABBB	BBABA
AABAB	ABABAAB	ABABBBB	BBABAA
AABBA	ABABAAA	ABBAAAA	BBABAB
			BBBA
			BBBAB
			BBBB

n = 15

B5

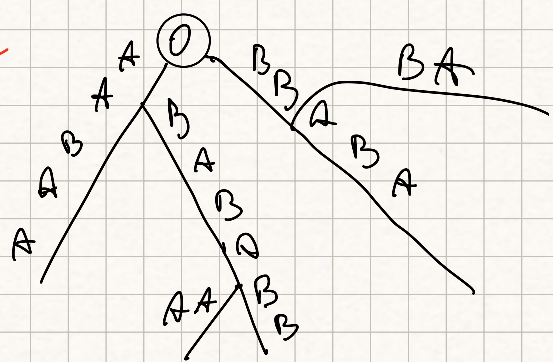
FC - storage per block

AABAA <4,B>, <3,BA>

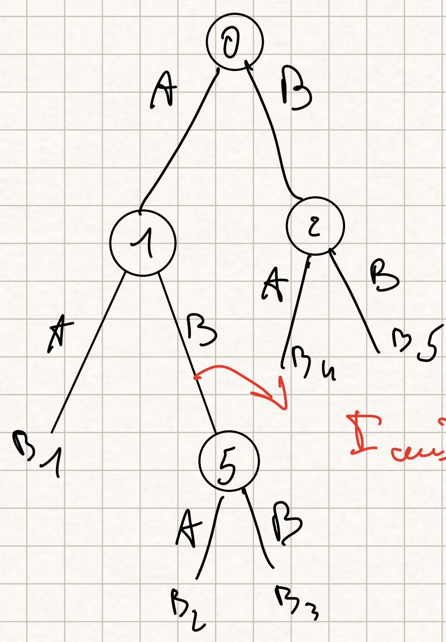
ABABAAA <6,B>, <5,BA>

	AABAA	ABABAAA	ABABABB	BBABA	BBBA
DISK:	B_1	B_2	B_3	B_4	B_5

uncompacted trie



Patricia Trie



I will search between

B_3, B_4

0 1

P = ABBAB

Searching (Dictionary problem)

- Exact match : hash
- prefix match : uncompacted / compacted tries
  Patricia trie
  array of pointers
- substring match : suffix array / tree

reduce substring to prefix search

Given a text $T[1,n]$ drawn from an alphabet Σ, preprocess T and build a d.s. so that, given on-the-fly a pattern $P[1,p]$ find all occurrences of P in T.

count

list retrieve

- $C = \{T_1, T_2, \dots, T_k\} \rightarrow T = T_1 \$ T_2 \$ T_3 \dots \$ T_k$

Reduction from substring to prefix match

$SUF(T) = \{T[i,n], \forall i = 1,2,\dots,n\}$ set of all suffixes of T

$T =$ (1 2 3 4 5 6 7) A B A C A B C

$SUF(T) =$
- A B A C A B C
- B A C A B C
- A C A B C
- C A B C
- A B C

$P = AB$

$\begin{cases} BC \\ C \end{cases}$

If P occurs in T at position $i \iff P = T[i, i+|P|-1]$

$\iff P$ is a prefix of $T[i, n]$

Suffix array: array of pointers to text suffixes sorted alphabetically

SA	
1	ABACABC
5	ABC
3	ACABC
2	BACABC
6	BC
7	C
4	CABC

$O(n)$ $O(n^2)$ space

Data structure: SA and T

$n \log n + n \log |\Sigma|$ bits

$T = AB\,ACA\,BC$

$P = AB$

AB$ smaller than any character

AB# larger than any character

$\$ < \text{any } \Sigma < \#$

$O(n \log n)$ bits = (n = pointers · $\log n$ = avg suffix len)

·) Count (P): prefix search from P$ and P#: $O(\frac{P}{B} \log n)$ I/Os

$O(P \log n)$ time

·) Retrieve (P): prefix search from P$ and P#: scan the subarray between positions of P$ and P# and print

$O(P \log n + occ)$ time

$O\left(\frac{P}{B} \log n + \frac{occ}{B}\right) I/Os$

binary search pfixes

SA_build(T, n):

```
for (i=1; i<=n; i++)
    SA[i] = T+i-1;
```

} $O(n)$

```
qsort(SA, n, sizeof(char*), suffix_cmp);
```

} $O(n^2 \cdot n)$ on avg $O(n(n \log n))$

```
return SA;
```

Suffix_cmp(char **a, char **b):

```
return strcmp(*a, *b);
```

$O(n(n \log n)) = O(n^2 \log n) = O(c \cdot n \log n)$

↑ avg # of compared chars

$T = A B B B C B B A$

1 2 3 4 5 6 7 8

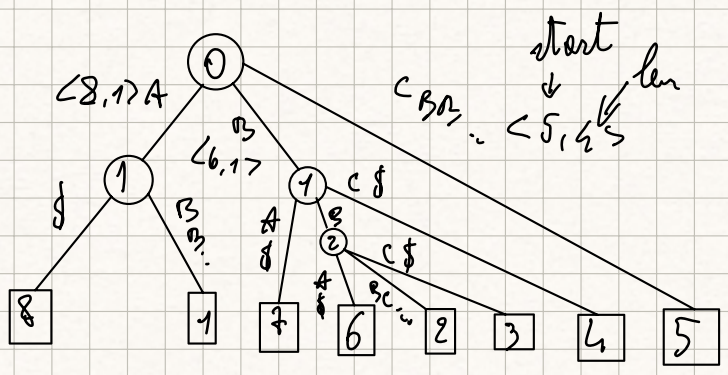
Longest common prefix $[1, n-1]$

$lcp[i]$ = longest common prefix (suffix $SA[i]$, suffix $SA[i+1]$) for $i < n$

LCP	SA	
	8	A $
1	1	A B B B C B B A $
0	7	B A $
1	6	B B A $
2	2	B B B C $
2	3	B B C $
1	4	B C $
0	5	C B B A $

Suffix tree = compacted trie of all suffixes of the string $T[1, n]$

$SA + LCP \rightarrow ST$



n^2

In a compacted trie the number of edges is proportional to the number of leaves

leaves = n

int. nodes $< n$

edges $< n$

$\Big\}$ $O(n)$ space

with in order search we obtain the LCP.

with the leaves we obtain the SA.

1) Check if it exists a substring of T that repeat ≥ 2 time

We use LCP. We check ST

2) Check if it exists a substring of T that repeat ≥ 2 time and has len $= L$

Check if an internal node of ST has value $\geq L$

Check if LCP array has a value $\geq L$

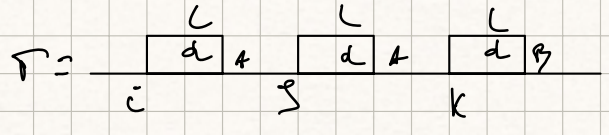
3) Check if it exists a substring of T that repeat $\geq C$ time and has len $= L$

In ST we search for a node with value $\geq L$ and a number of child $\geq C$

In LCP array: $\exists\ LCP[i, i+C-2] \geq L$

$C-1$

Dim :

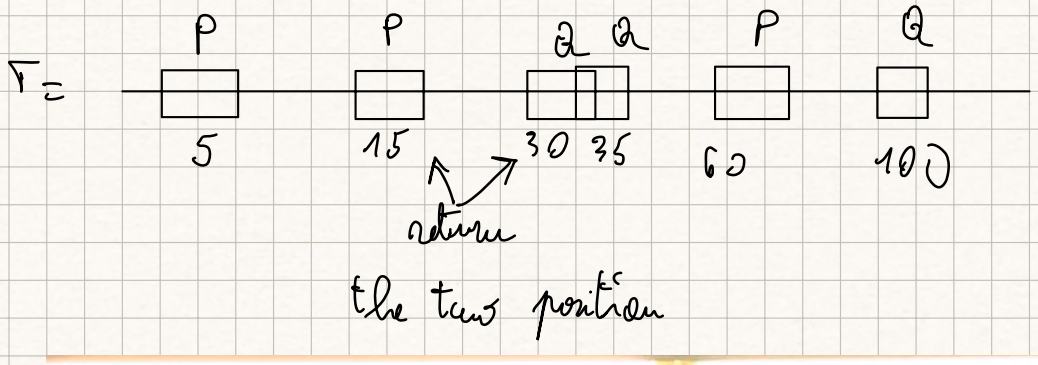


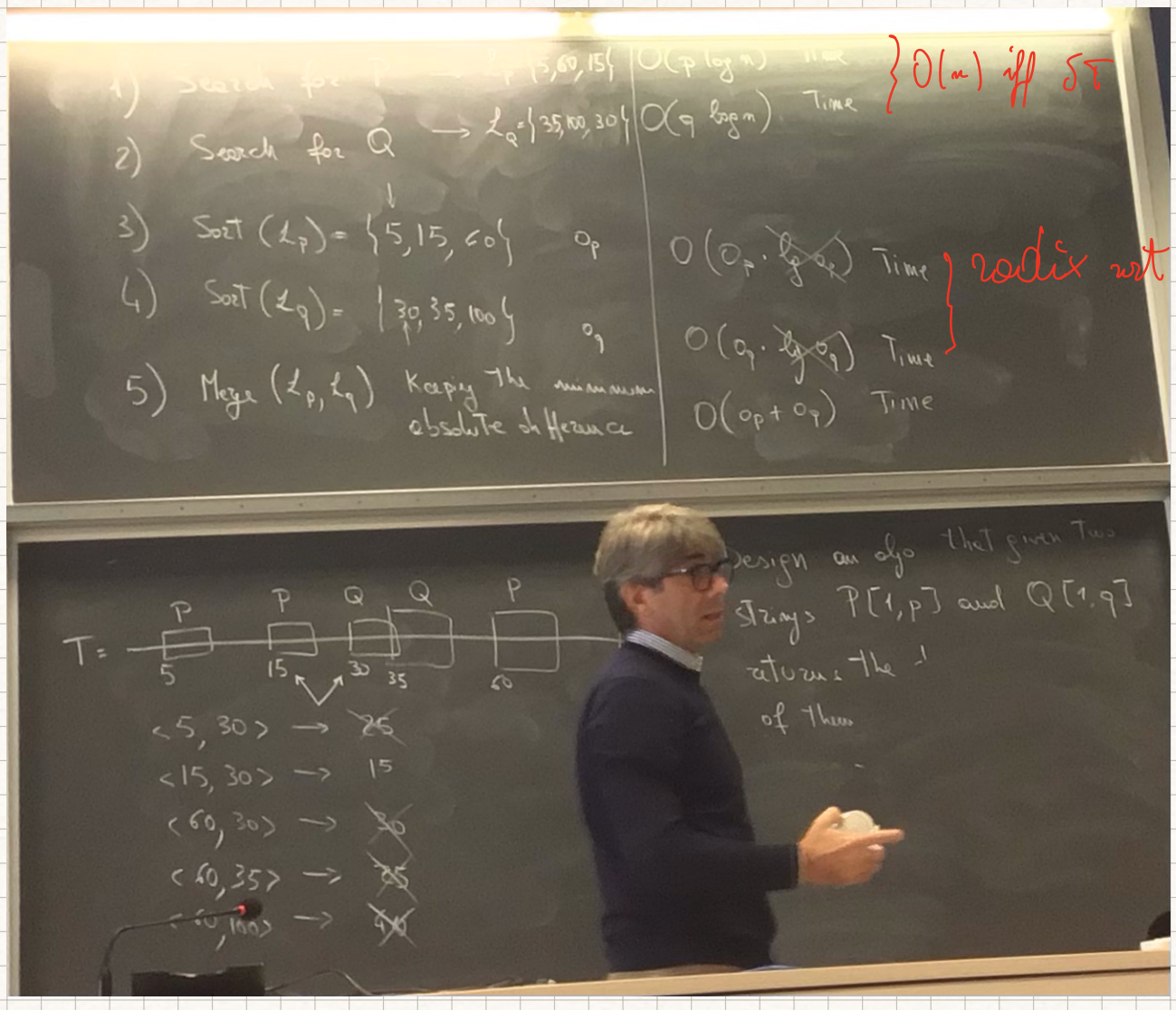
$C = 3$

$\geq L$

$\geq L$

Design an alg. that given two strings P[1, p] and Q[1, q] return the closest occurrence of them in a text T[1, n] Estimate the space and time complexity.





O(n) linear

Entropy

Source → emits symbols from an alphabet Σ

Probability p_σ to emit σ: $0 \le p_\sigma \le 1$ $\quad \sum_\sigma p_\sigma = 1$

information content of symbols σ

entropy of the source: $H_0 = \sum_\sigma p_\sigma \cdot \log_2 \frac{1}{p_\sigma}$ bits

$\forall \sigma' \neq \sigma$

$\neq 0$; $= 0$ iff $\begin{cases} p_{\sigma'} = 0 \\ p_\sigma = 1 \end{cases}$

$p_\sigma = \frac{1}{|\Sigma|}$ → equals probability → $\le \log_2 |\Sigma|$

H

$\log_2 \frac{1}{p_\sigma}$ max entropy

0 memory

measure the randomness of the source

$$E[X] = \sum_v \underbrace{P(X=v)}_{p_\sigma} \cdot \underbrace{v}_{\log_2 1/p_\sigma}$$

$X = \log_2 \frac{1}{p_\sigma}$ with prob. p_σ

prefix-free codes

a → 0
b → 10
c → 11

} codewords (unique decodability)

↓

codewords are not prefix of each other

$\Sigma = \{a, b, c\}$ $\quad \frac{1}{4} \quad \frac{1}{2} \quad \frac{1}{4}$

$H = p(a) \cdot \log_2 \frac{1}{p(a)} + p(b) \cdot \frac{1}{p(b)} + p(c) \log_2 \frac{1}{p(c)}$

$= \frac{1}{4} \cdot \log_2 4 + \frac{1}{2} \cdot \log_2 2 + \frac{1}{4} \cdot \log 4 = \frac{1}{2} + \frac{1}{2} + \frac{1}{2} = 1{,}5$ bits

Average codeword length $= \sum_\sigma p(\sigma) \cdot |cw(\sigma)| =$

$= p(a) \cdot 1 + p(b) \cdot 2 + p(c) \cdot 3 =$ bits

$= \frac{1}{4} + \frac{1}{2} \cdot 2 + \frac{1}{4} \cdot 3 = 1 + \frac{1}{4} + \frac{3}{4} = 2$ bits/symbols

Th (Shannon): any prefix-free code for some source takes on average # bits per symbol $\geq H$ (entropy)

$\Sigma = \mathbb{N}$ $\quad X \in \mathbb{Z} \to \begin{cases} 2|x|+1 & \text{if } x < 0 \\ 2x & \text{if } x \geq 0 \end{cases} \in \mathbb{N}$

$S = s_1 s_2 s_3 \dots s_m$, $s_i \in \mathbb{N}$, s_i may repeat

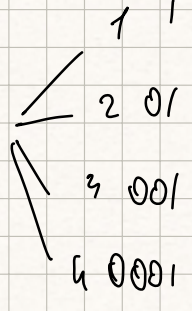
$S' = s_i < s_j$ $\quad i < j$ increasing sequence

for-encoding 1 2 11 3 ...

$b = \lceil \log_2 (S^*+1) \rceil$ bits $\quad S^* = 12 \quad S^*+1 = 13$ fixed-length codeword code

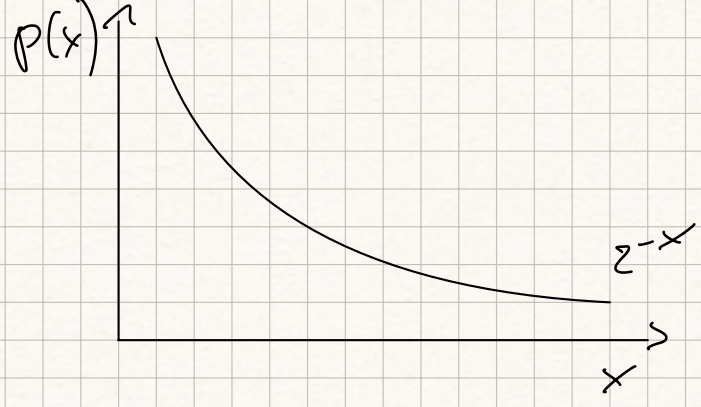
(S^* = max s_i)

Theory: $\forall x \geq 1$ $\quad \underbrace{0 \dots 0}_{x-1} 1 = 0^{x-1}1$



if $|Cw(\sigma)| = \log_2 \frac{1}{P(\sigma)}$ code is optimal

$|U(x)| = \log \frac{1}{P(x)} \iff x = \log_2 \frac{1}{P(x)} \iff 2^x = \frac{1}{P(x)} \iff P(x) = 2^{-x}$



Elias:

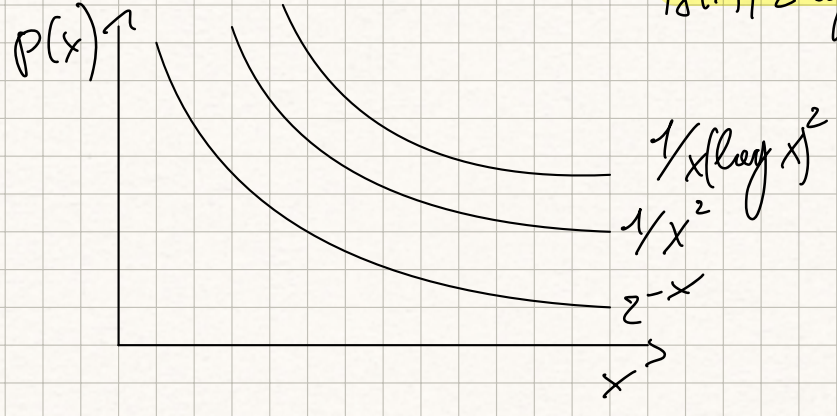
γ-code : $\gamma(x) = \underbrace{0\dots0}_{\ell-1}\ \underbrace{bin(x)}_{\ell}$ $\quad |\gamma(x)| = 2\ell - 1 = 2\log_2 x$

$\gamma(5) = 00\,101$

$|U(x)| = \log \frac{1}{P(x)} \iff 2\log_2 x = \log_2 \frac{1}{P(x)} \iff P(x) = \frac{1}{x^2}$ Power law

δ-code : $\delta(x) = \underbrace{\gamma(\ell)}_{2\log \ell - 1}\ \underbrace{bin(x)}_{\ell}$ $\quad \delta(5) = \gamma(3)\,101 = 011\,101$

$|\delta(x)| = 2\log_2(\log x) - 1 + \log(x)$ bits



$2\log(\log x) + \log x = \log_2 \frac{1}{P(x)}$

$\log(\log x)^2 + \log x = \log_2 \frac{1}{P(x)}$

$\log^2(x) \cdot x = \frac{1}{P(x)}$ $\quad P(x) = \frac{1}{\log^2(x) \cdot x}$

Variable byte :

x^0 : 00 1 ...

$$\left\lceil \frac{|bin(x)|}{7} \right\rceil \cdot 8 \text{ bits}$$

tagging: add bit '1' in front of each block, '0' for the last one

$bin(x) =$ 1 0000000 1 0001011 0 1100100

padding | 7 | 7 | tagging

Fast Delta: blocked scheme

Two parameter: base, b

$x \in S : x \geq base$

partition S in two subsets

- $x \in [base, base + 2^b - 2] = S_1$
- $x \geq base + 2^b - 1 = S_2$

$x - base \geq 0$

- $x \in S_1$: represent x in b bits
- $x \in S_2$: write $\underbrace{11\ldots1}_{b} \Rightarrow 2^b - 1$ and then encode x in 4 bytes

exception

$S = 0 \quad 1 \quad 6 \quad (7) \quad 3 \quad 3 \quad 9 \quad 2$

000 001 110 111 011 011 111 010

$b_{max} = 3$

$b = 3$ — $[0,6]$

≥ 7

Take b such that $\geq 90\%$ of numbers in S are in S_1

Rice code (parameter k)

$x \geq 1$ $R_k(x)$

- $q = \left\lfloor \frac{x-1}{2^k} \right\rfloor$ quotient → encoded as $U(q+1)$
- $r = x - 1 - 2^k \cdot q$ remainder → encoded as $bin_k(r)$ (k bits)

$$|R_k(x)| = \left\lfloor \frac{x-1}{2^k} \right\rfloor + 1 + k$$

$$R_4(20) = \begin{cases} q = \left\lfloor \frac{19}{16} \right\rfloor = 1 \\ r = 3 \end{cases}$$

$U = 01$

$bin_4(3) = 0011$

$$R_3(20) = \begin{cases} q = \left\lfloor \frac{19}{8} \right\rfloor = 2 \\ r = 3 \end{cases}$$

$U = 001$

$bin_3(3) = 011$

optimal distribution $P(x) = p(1-p)^{x-1}$

Elias-Fano code

$S = s_1' \; s_2' \; \dots \; s_n'$ increasing (distinct)

universe $= s_n' + 1$ $\quad b = \lceil \log_2 u \rceil$ $\quad \ell = \lceil \log_2 \frac{u}{n} \rceil$ (lower) $\quad h = b - \ell$

(ℓ: lower; # of numbers)

1	00001
4	00100
7	00111
18	10010
24	11000
26	11010
30	11110
31	11111

h | ℓ

$L = 01\,00\,11\,10\,00\,10\,10\,11$ $\quad |L| = n \cdot \ell = n \cdot \lceil \log \frac{u}{n} \rceil$

$H = \underbrace{10}_{0}\,\underbrace{110}_{1}\,\underbrace{0}_{2}\,\underbrace{0}_{3}\,\underbrace{10}_{4}\,\underbrace{0}_{5}\,\underbrace{110}_{6}\,\underbrace{110}_{7}$ $\quad |H| \le 2n$

(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16)

negated unary representation for (size + 1) of every configuration

$h = 3$

$b = 5$ bits

$\ell = 2$

$n = 8$

(1)

1 in $H = n$

0 in $H = 2^h$
- # 0 is at most $n + 1$
- in this example $2^h = n$

Represent H up to the last 1

(2) # 0 in $H \le n$ (dropping the last one)

Total size $= n(\lceil \log_2 \frac{u}{n} \rceil + 2)$ bits

Max $S = 18$

$u = 18 + 1 = 19$ $\qquad$ $u = 8$

$b = \lceil \log_2 u \rceil = 3$ bits $\quad l = \lceil \log_2 \frac{u}{n} \rceil = \lceil \log_2 \frac{19}{8} \rceil = 2$

$h = 3$

1	000	01
2	000	10
3	000	11
4	001	00
5	001	01
6	001	10
7	001	11
18	100	10

Access the i-th number:

① pick the i-th group of l bits in L

② $y = \text{select}_1(i) - i$, represent y in h bits

$\text{select}_1(i)$: return position of the i-th 1 in H

Access(3) = 001|11 $\quad y = \text{select}_1(3) - 3 = 4 - 3 = 1$

3-th compressed elem

Access(6) = 110|10 $\quad y = \text{select}_1(6) - 6 = 12 - 6 = 6$

$\text{Rank}_1(i)$, $\text{Rank}_0(i)$: counting how many 1/0 are in the list until i-th pos.

$\text{Select}_1(i)$, $\text{Select}_0(i)$: positions of i-th 0/1

GEQ(i) // greater or equal than i

```
p = select_0(H(i)) + 1;
s = p - H(i);
repeat:
    y = Access(s);
    s++;
while (y ≥ i) && (s ≤ n)
```

high part of bin(i)

return the position of H(bin(i)) zeros in H of Elias-Fano

ottengo la posizione del primo elemento del bucket corrispondente alla H(bin(i)) configurazione.

ottengo l'indice dell'elemento del bucket della configurazione H(bin(i)) del primo

es: 0, 3, 5, 8, 9, 10, 11

Access(5) = 9

GEQ(6) = 8

$b = \lceil \log_2 12 \rceil = 4$

$l = \lceil \log_2 \frac{12}{7} \rceil = 1$

h = 3

0	000
0	011
0	101
1	000
1	001
1	010
1	011

L = 0 1 1 0 1 0 1

H = 1 0 1 0 1 0 0 1 1 0 1 1 0

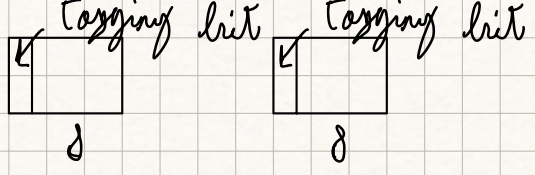
Access(5) => select$_1$(5) − 5 = 9 − 5 = 4 => 1001

GEQ(6) => 6 = 0110 p = select$_0$(3) + 1 = 7

s = 7 − 3 = 4

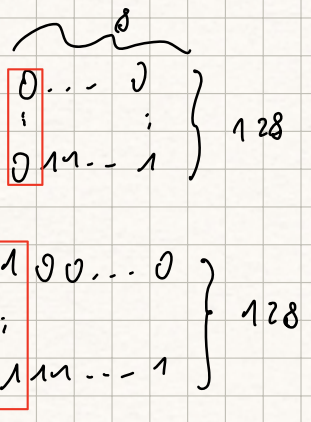
Access(4) => select$_1$(4) − 4 = 4 1000

(S, C) - dense code ← Variable byte



·) S + C = 2^b = # configuration on b bits

·) b = 8 bits ⟶ # Config = 256 { S stoppers, C containers



if b = 3 bits ⇒ 2^3 = 8 Config

S + C = 8

S = C = 4 → 000, 001, 010, 011 } S ; 100, 101, 110, 111 } C

S = 6, C = 2 → 000, 001, 010, 011, 100, 101 } S ; 110, 111 } C

number	codeword (4,4)	codeword (6,2)	codeword (1,7)
0	000	000	000
1	001	001	001 000
2	010	010	010 000
3	011	011	011 000
4	100 000	100	100 000
5	100 001	101	101 000
6	100 010	110 000	110 000
7	100 011	110 001	111 000
8	101 000	110 010	

C C S

001 001 000

001 010 000

Interpolation code (sequence increasing)

l r

1, 2, 3, 5, 7, 9, 11, 13, 18, 19, 20, 24

1, 1, 2, 2, 2, 2, 4, 3, 1, 1, 1 → 1° gap-encoding

parameters: l, r, low, high (positions, values)

l = 1, r = 12, low = 1, high = 21

position

$m = \left\lfloor \frac{l+r}{2} \right\rfloor = 6 \rightarrow S_6 = 9$

using the minimum # bits

1, 2, 3, 5, 7, 9, 11, 13, 18, 19, 20, 24

l m r

9 | 18

left recursion call: l, m-1, low, high → <1, 5, 1, 8>

right recursion call: <7, 12, 10, 21>
m+1 r low high
l

$Q = low + m - l \leq 9 \leq high - r + m = b$

$Q = 1 + 6 - 1 = 6$ $\quad b = 21 - 12 + 6 = 15$

$m = 6, S_m = 9 \rightarrow low + m - l = 6 = Q$
1 6 1

$\rightarrow high - r + m = 15 = b$

encode $S_m - a = 3$ = 0011 using $\lceil \log_2 15 - 6 + 1 \rceil = \lceil \log_2 10 \rceil = 4$ bits ($b - a + 1$)

left $< l, 2, low, high > \Rightarrow$ left $< l, m-1, low, S[m] - 1 >$
1 5 1 8

$m = 3$, $S_m = 3$ $\left\lceil \frac{l+2}{2} \right\rceil = m$

$a = 1 + 3 - 1 = 3$ $low + m - l$

$b = 8 - 5 + 3 = 6$ $high - a + m$

encode $[S_m - a] = 3 - 3 = 0$

in $\lceil \log_2 (6 - 3 + 1) \rceil = 2$ bits $\Big\}$ $(00)_2$

right $< l, r, low, high > \Rightarrow$ right $< m+1, 2, S[m] + 1, high >$
7 12 10 21

$m = 4$ $S_m = 18$

$a = 10 + 4 - 7 = 12$ $low + m - l$

$b = 21 - 12 + 4 = 18$ $high - a + m$

encode $S_m - a = 18 - 12 = 6$

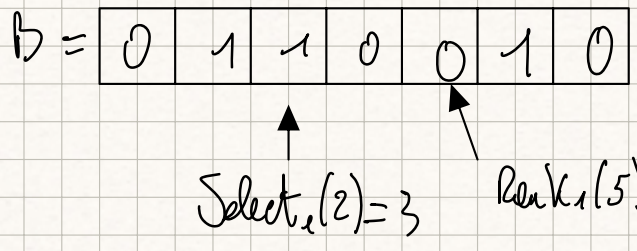
in $\lceil \log_2 (18 - 12 + 1) \rceil = 3$ bits $(110)_2$

Compressed/compacted d.s.

$Rank_b(i)$ = count in $B[1,i]$ the number of b bits

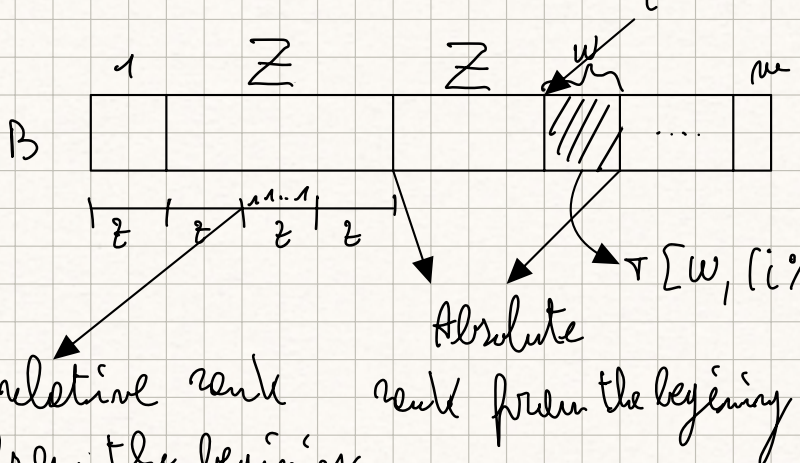
$Select_b(i)$ = position of the i^{th} bit $= b$

$Rank_1(i) = i - Rank_0(i)$



Th 1: $\exists$ data structure that occupies $O(m)$ bits $\left(\simeq \frac{m}{\log_2 m}\right)$ in addition to $B[1,m]$ and support $Rank_1$ in $O(1)$ time

~~$Rank_1$~~ / $Select_1$ / $Select_0$



Big blocks $= \frac{m}{z}$

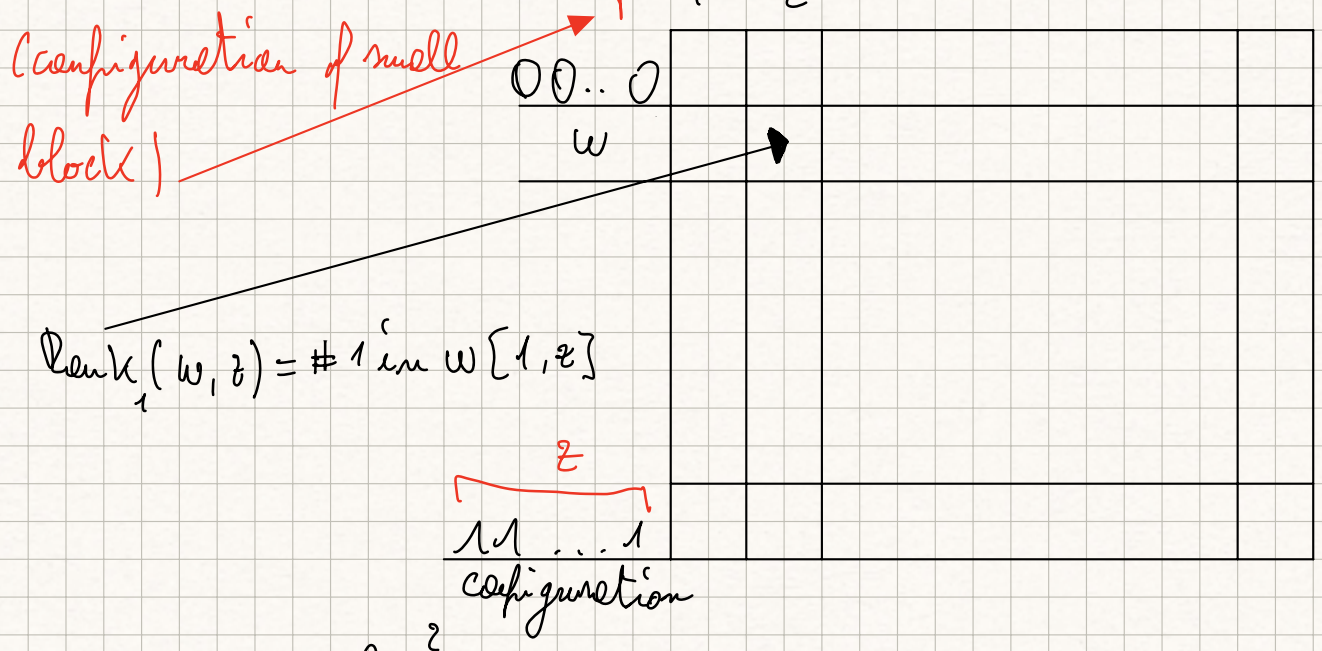
Small block $= \frac{m}{t}$

Time $= O(1) + O(z)$

Space in bits $= O\left(\frac{m}{z} \cdot \log m + \frac{m}{t} \cdot \log z + t \cdot 2^t \log z\right)$

space of absolute rank of big blocks | space of relative rank of small blocks | space of lookup table T

Four-russian Trick



We set: $Z = \log^2 n$, $z = \frac{1}{2}\log n$

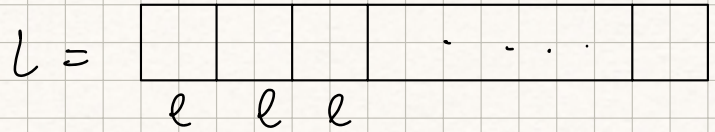
Big zeta, small zeta

Then, the space:

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

$$O\left(\underbrace{\frac{n}{\log n} + \frac{n}{\log n}\log\log n}_{(n\log\log n/\log n)} + \underbrace{(\log n)\sqrt{n}\,(\log\log n)}_{n^{1/2+\varepsilon}}\right) = O(n)$$ ■

Alice - Four idea to compress



$n\lceil\log_2 \frac{u}{n}\rceil$ bits

Th: $H = \underbrace{11101100\ldots}_{2n \text{ bits}}$ $\left.\begin{matrix}\text{Select}_1\\ \text{Select}_0\end{matrix}\right\}$ $+ O(n)$ bits

Th 2: Elias-Fano for encoding an integer array can be implemented in $2n + n\left\lceil \log_2 \frac{u}{n} \right\rceil + O(n)$ bits and support Access in $O(1)$ time and nextGeq in $O(\log \frac{u}{n})$ time

$$O\left(n\left(2 + \left\lceil \log_2 \frac{u}{n} \right\rceil + o(1)\right)\right)$$

Given $B[1, m]$, construct $A[1, n]$ such that $A[i]$ = position of i-th 1.

	1	2	3	4	5	6 = m
B =	0	1	1	0	1	1

$O(m)$ $n = 4$

$\Downarrow$

	1	2	3	4 = n
A =	2	3	5	6

$O(n)$

Th 1 $\left\{\begin{matrix} m + o(m) \text{ bits} \\ O(1) \text{ rank/select}\end{matrix}\right.$

We build A putting the positions of 1 occurring in B.

build Th 2 over A → Elias-Fano + select over array H

(m = max pos in B)

$\left\{\begin{matrix} n\left(2 + \left\lceil \log \frac{u}{n} \right\rceil\right) + O(1) \text{ bits} \\ O(1)\end{matrix}\right.$ Th 2

Access $O(1)$

GGQr $O(\log \frac{m}{n})$ bits

$\text{Select}_{1,B}(i) = \text{Access}_{A}(i)$ in $O(1)$ time

$\text{Rank}_{1,B}(j) = GGQ_{A}(j+1) - 1$ in $O(\log \frac{m}{n})$ time

$S =$

B	O	X	\0	C	A	T	\0	P	A	O	L	O	\0	...	

m

$A =$

1	5	9	...

$B =$

1	0	0	0	1	0	0	0	1	

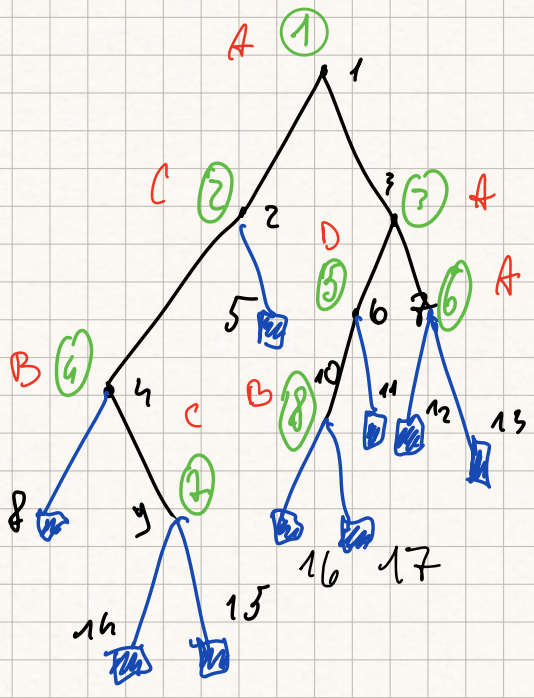
Build Sol 1 compact

- $D + o(D)$ bits
- Retrieve a string i-th: Scan S from position : $\text{Select}_1(B, i)$

Build Sol 2 compressed

- $n\left(2 + \log \frac{D}{n} + O(1)\right)$

Binary tree: Succint representation of binary trees (pointerless programming)



① tree completition
- add leafs
- add dummy nodes

② node labeling (breadth-first)

③ serialization (1 if the position correspond to an existing node)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
B =	1	1	1	1	0	1	1	0	1	1	0	0	0	0	0	0	0
	m=1	2	3	4		5	6		7	8							

$= m$ + $Rank_1$ $Select_1$

Total space occupancy: $B + o(|B|)$ $\quad |B| = m = 2n+1$ bits

$n[2+o(1)]+1 = 2n+1+o(n) = n+o(n)$ bits $\quad O(1)$ time

only 2 bits per node

nodes

$2^{2n} \geq \log_2 2^{2n} \geq 2n$ bits to represent a tree

left_child(x) = $Rank_1(2x)$ → skip the fake nodes (leafs) if $B[2x]=1$

not circled

right_child(x) = $Rank_1(2x+1)$ if $B[2x+1]=1$

$$parent(\otimes) = \left\lfloor \frac{select_1(X)}{2} \right\rfloor$$

$L =$

A	C	A	B	D	A	C	B
1	2	3	4	5	6	7	8

Data compression:

'50-'60. Statistical compressors (Huffman, Aritmetic, ...)

'70. Dictionary compressors (Lempel-Ziv family < LZ77 / LZ78)

'80. Block sorting compressors (Burrows-Wheeler transform ...)
↓
gzip, 7zip

Compressed-file: | PREAMBLE | BODY |

PREAMBLE: Σ, prob

BODY: 01001 ...

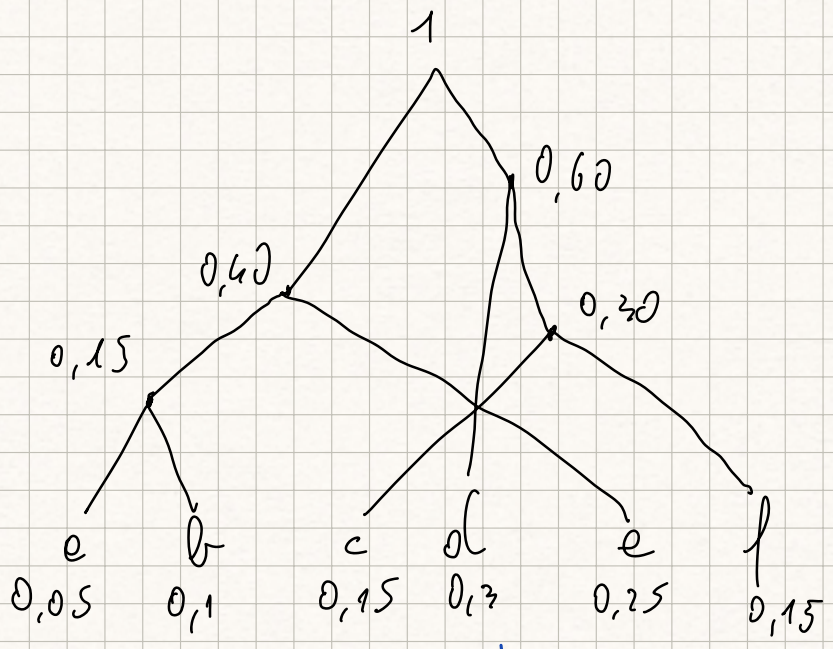
We discuss this in the following lecture

$\Sigma = \{a, b, c, d, e, f\}$ sum = 1

a	b	c	d	e	f
0,05	0,1	0,15	0,3	0,25	0,15

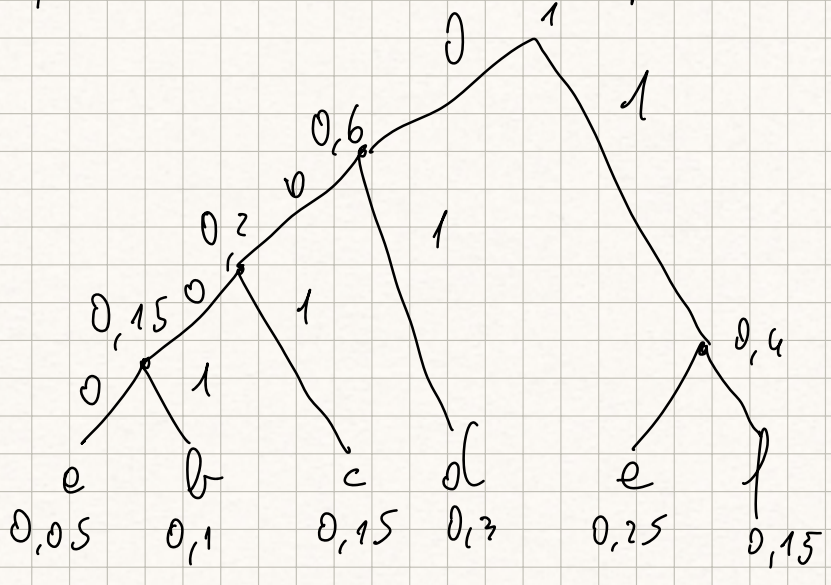
Huffman: ① pick the two nodes with smallest probability
② merge them into a parent node

③ enjoy to percent the sum of prob.



Huffman tree

encoding of e will be 0000, the path from root to leave



Th: Huffman code is optimal among all prefix-free code for alphabet Σ

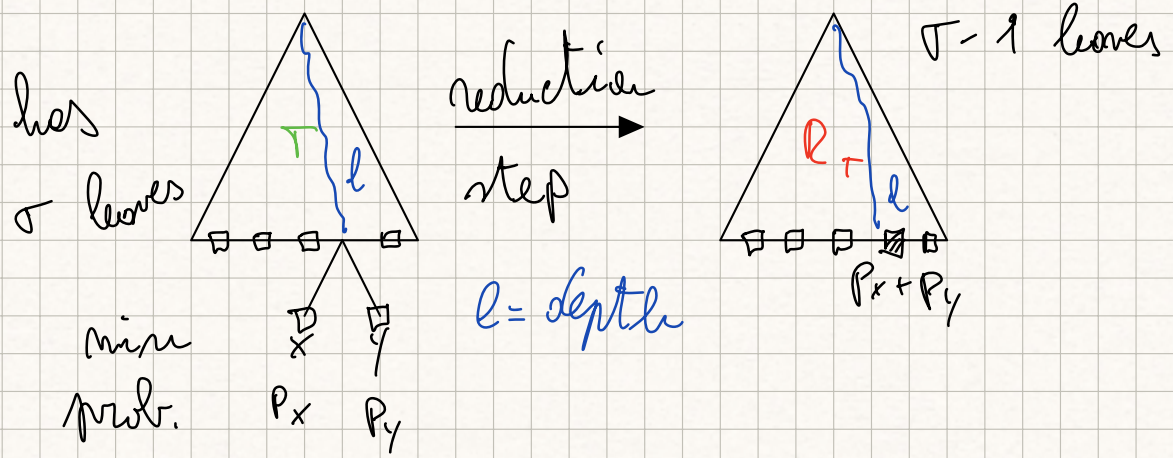
$\Updownarrow$

$L_H \leq L_C \quad \forall C$ prefix-free code

Lemma 1: F = set of binary trees with Σ leaves where avg depth is minimum. $\exists$ tree $\in F$ such that the two leaves with min prob. are at the largest depth and are children of the same parent.

avg depth of a leaf = avg codeword length $L_C = \sum_{s \in \Sigma} p(s) \cdot |cw(s)|$

Lemma 2:



ℓ = depth

L_T = average depth tree T $\sigma \neq x, y$

$$L_T = \sum_{\sigma} p(\sigma) \cdot \ell(\sigma) = \sum_{\sigma \neq x,y} p(\sigma)\ell(\sigma) + p(x)\ell(x) + p(y)\ell(y) =$$

$$= \sum_{\sigma \neq x,y} p(\sigma)\ell(\sigma) + p(x)(\ell+1) + p(y)\cdot(\ell+1) =$$

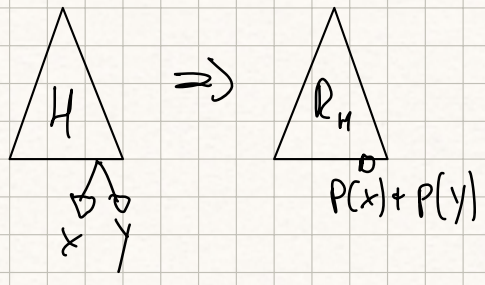
$$= \sum_{\sigma \neq x,y} p(\sigma)\cdot\ell(\sigma) + [p(x)+p(y)]\cdot(\ell+1)$$

$$L_R = \sum_{\sigma \neq x,y} p(\sigma)\cdot\ell(\sigma) + \ell(p(x)+p(y))$$

$$L_T - L_Q = (\ell+1)\cdot p(x) + (\ell+1)\cdot p(y) - \ell(p(x)+p(y)) = p(x)+p(y)$$

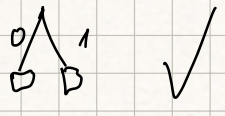
$$L_T = L_Q + p(x) + p(y)$$

Dim Huffman th:



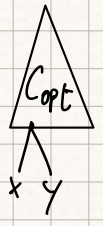
$$L_H \overset{②}{=} L_{R_H} + p(x) + p(y)$$

Proof by induction:

$\sigma = 2$  ✓

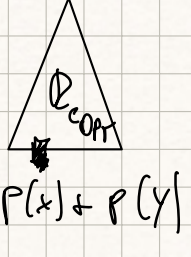
Huffman is optimal for an alphabet of $\sigma-1$ symbols

let C_{opt} an optimal code for σ symbols ⇒ avg depth is $L_{C_{opt}}$



lemma 1 ↙ leaf with the smallest prob. are at the deepest depth

lemma 2 ⇓



$$L_{C_{opt,\sigma}} = L_{C_{opt,\sigma-1}} + p(x) + p(y)$$

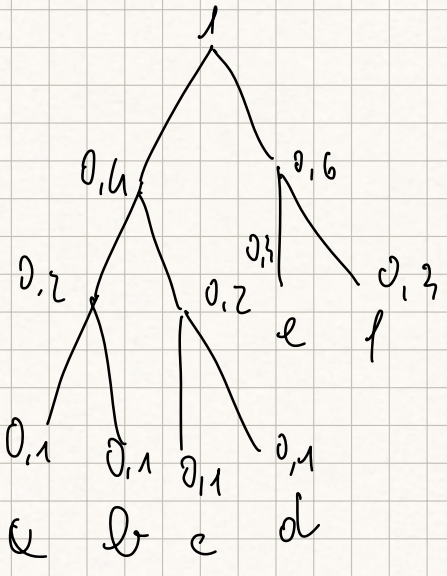
$$L_{H,\sigma} \overset{②}{=} L_{R_{H,\sigma-1}} + p(x) + p(y) \overset{\text{hypothesis}}{\leq} L_{R_{C_{OPT,\sigma-1}}} + p(x) + p(y) \overset{②}{=} L_{C_{OPT,\sigma}}$$

optimal for $\sigma-1$ symbols according to the inductive hyp.

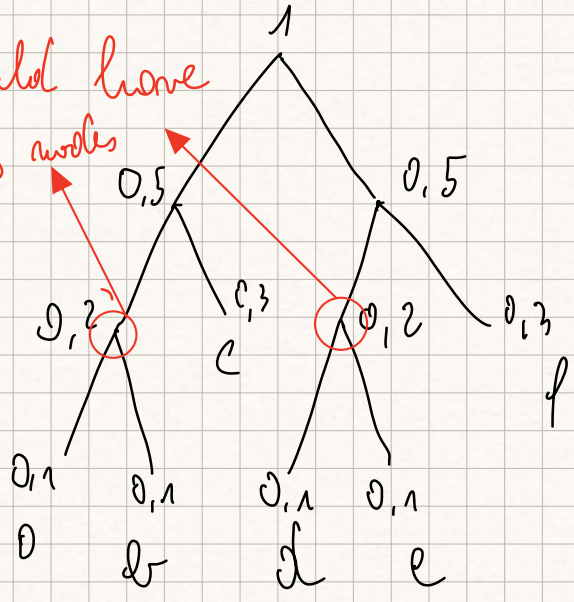
smallest prob. over σ symb.

$$L_{H,\sigma} \leq L_{C_{OPT},\sigma}$$ ■

Huffman tree is shallowest than the other



we should have merged this nodes



Huffman tree

NO Huffman tree

entropy

$$H_{entropy} \leq L_{Huff} < H + 1 \text{ bits}$$

Shannon

avg codeword length

every codeword on avg looses < 1 bit w.r.t. entropy

$$0 \leq H \leq \log_2 |\Sigma|$$

if we have a repetitive source of symbols (skewed distribution)

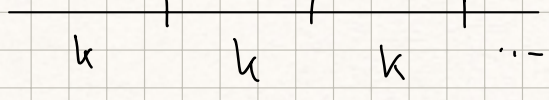
HUFFMAN init the best choices

$H \longrightarrow 0$ $\quad L_{Huff} \geq 1$ bit skewed distribution

$H \longrightarrow \log_2 |\Sigma|$ $\quad L_{Huff} \simeq H$ uniform distribution

to reach the entropy we accorporate data together in order to form the macro symbols

$|\Sigma^1| = |\Sigma^4|$



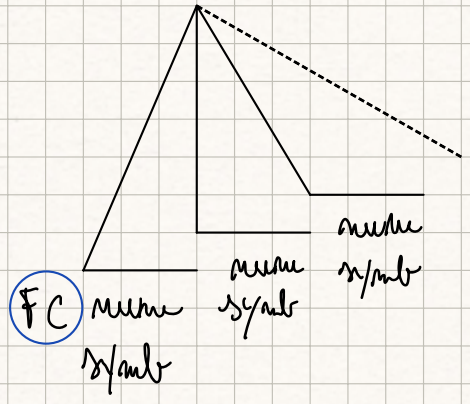
avg cw length

$$H \leq \frac{L_{Huff}}{k} \text{ for a block of length } k < H + \frac{1}{k}$$

1948

avg number of bits per symbol taken by a cw

Input: Huffman tree ⟶ Canonical Huffman tree



First codeword

given num[1, maxcw] : array of ints

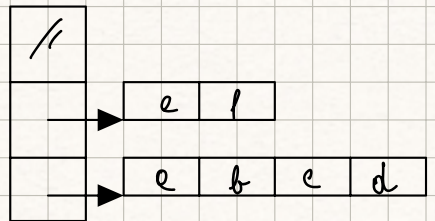
symb[1, max cw] : array of arrays of symbols

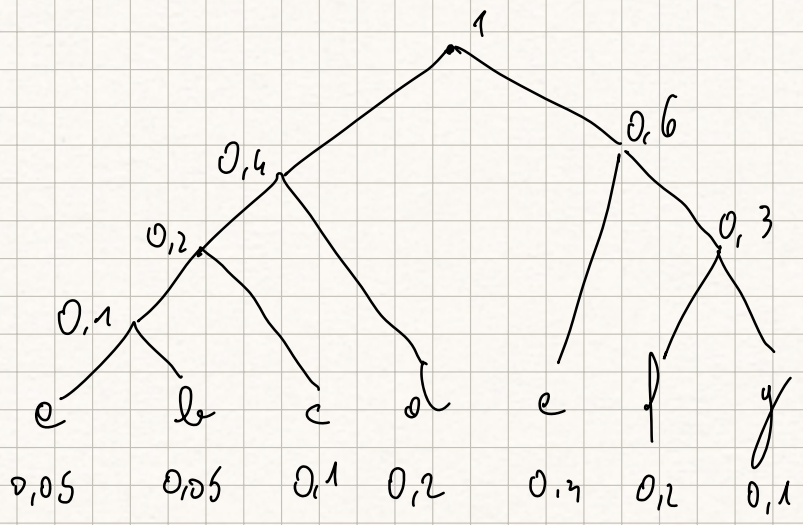
num[1, 3] = [0, 2, 4]

1 2 3

number of symbols per level

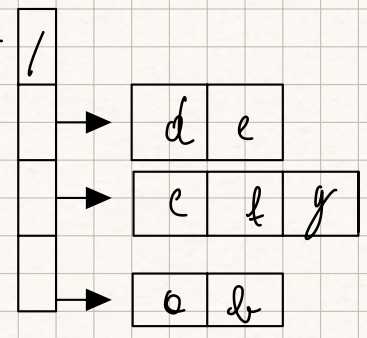
symb[1, 3] :





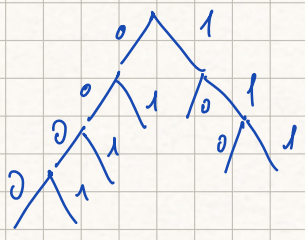
num [1,4] = [0, 2, 3, 2]

w/mb:



FC [max level] = 0

4



$$FC[i] = \frac{FC[i+1] + num[i+1]}{2}$$

$$FC[3] = \frac{FC[i+1] + num[4]}{2} = \frac{0+2}{2} = 1$$

$$FC[2] = \frac{FC[3] + num[3]}{2} = \frac{1+3}{2} = 2$$

$$FC[1] = \frac{FC[2] + num[2]}{2} = \frac{2+2}{2} = 2$$

FC [1,4] = [2, 2, 1, 0]

1 2 3 4

Compressione: 0 1 0 0 0 0 1 1 1 → f b e

• $\left(FC\left[\text{level symb}\right] + \text{offset}(\text{symb})\right)_2$

Pseudocode:

Decompressor ():

v = next_bit(); // prendo il primo bit della sequenza

l = 1; // parto dal primo livello

while [v < FC[l]) // finché $(v)_{10} < (FC[e])$ vuol dire che non sono arrivato al livello in cui è presente il simbolo

v = 2·v + next_bit(); // eseguo cambio di base di v 2 → 10

l++ // incremento livello

return symb [l, V - FC[l]] // decomprimo e ottengo il simbolo

f: v=0, l=1

v=1, l=2

v=2, l=3

symb[3, 2-1]

b: v=0, l=1; v < FC[1] = ?

v = 2·0 + 0, l=2; v=0 < FC[2] = 2

v = 2·0 + 0, l=3; v=0 < FC[3] = 1

v = 2·0 + 1 = 1, l=4; v=1 > FC[4] = 0 STOP

symb [4, 1-0] = symb [4, 1] = b

e: v=1, l=1; v=1 < FC[1] = 2

V = 2·V + next_bit() = 3, l=2;

V=3 > FC[2] = 2

symb [2, 3-2] = e

$$\text{compression ratio} = \frac{\text{len compr. file}}{\text{len orig. file}} = \frac{9}{24} \leq 1$$

file of n symbols $\xrightarrow{\text{best}}$ n bits for the compr. file

$$C.R. = \frac{1n \text{ bits}}{8n \text{ bits}} = \frac{1}{8} \sim 15\%$$

Arithmetic Coding:

Converter(x), $x \in [0, 1) \longrightarrow .\, b_1 \, b_2 \, b_3 \dots$ $\quad b_i \in \{0, 1\}$

$\qquad\qquad\qquad\qquad\qquad\quad 1/2 \;\; 1/4 \;\; 1/8$

repeat until k bits are emitted

```
x = 2·x;
if x < 1 then output 0;
else {output 1;  x = x - 1;}
```

$x = 1/3 \rightarrow 2/3$	0
$4/3$	1
$1/3$	

$x = 1/3 \downarrow .01$

dyadic fraction $\quad x = \dfrac{v}{2^t}$, $\quad v \in \mathbb{N}$, $t \in \mathbb{N}$ $\qquad x = .\underbrace{00\dots0\ \text{bin}(v)}_{t \text{ bits}}$

$$\frac{5}{32} = \frac{5}{2^5} = .\,\boxed{0}\,\boxed{0}\,\boxed{1}\,\boxed{0}\,\boxed{1}$$

$\qquad\qquad\qquad\qquad 1/8 \qquad 1/32$

Lemma: given $x \in [0,1)$, if we truncate x to the first d bits $\hat{x}$ it holds $0 \le x - \hat{x} \le 2^{-d}$ error

$$x = .b_1 b_2 \dots b_d \, | \, b_{d+1} \dots$$
$$\hat{x} = .b_1 b_2 \dots b_d \, | \, 0 \; 0 \dots \Big\} \; 0 \le x - \hat{x} \le 2^{-d}$$

$$x - \hat{x} = .0 \dots 0 \, b_{d+1} \, b_{d+2} \le \underbrace{0 0 \dots 0}_{d} \, 1 1 1 \dots = \sum_{i=1}^{+\infty} 1 \cdot 2^{-(d+i)} =$$

$$= 2^{-d} \cdot \underbrace{\sum_{i=1}^{+\infty} 2^{-i}}_{1} = 2^{-d} \quad \blacksquare$$

$T = a \; b \; a \; c$ semi-static model $p(a) = \frac{2}{4} = \frac{1}{2}$ $p(b) = p(c) = 1/4$

$S_0 = 1$ $\quad S_i = S_{i-1} \cdot p(T[i])$

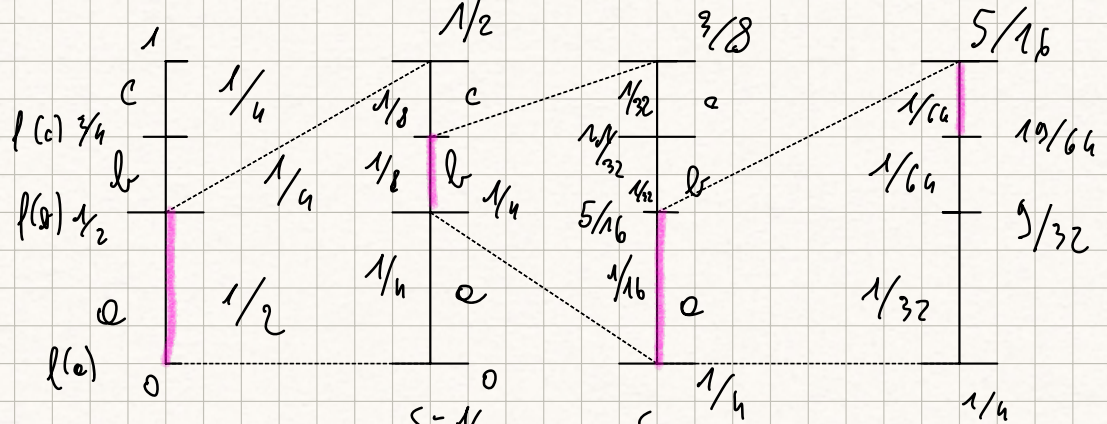
$l_0 = 0$ $\quad l_i = l_{i-1} + S_{i-1} \cdot p(T[i])$

$f(a) = 0$

$f(b) = p(b) = 1/2$

$f(c) = p(b) + p(c) = 3/4$

$$f(T) = \sum_{c < a} p(c)$$



$S_0 = 1$

$S_1 = 1/2$
$= S_0 \cdot p(a)$
$l_1 = 0$

$S_2 = 1/8$
$= S_1 \cdot 1/4$
$p(b)$
$l_2 = 1/4$

$S_3 = S_2 \cdot p(a) = 1/16$
$l_3 = 1/4$

$S_4 = 1/64$
$l_4 = 19/64$

$T = a \; b \; a \; c \longrightarrow \left[\frac{19}{64}, \frac{5}{16} \right)$

$\begin{cases} y = \text{pick a number inside} \\ n = \text{length of } T \end{cases}$

$s_0 = 1$; $l = 0$;

for $i = 1$ to n

$s_i = s_{i-1} \cdot p(T[i])$; ← probabilité del simbolo i-th della sequenza

$l_i = l_{i-1} + s_{i-1} \cdot f[T[i]]$; ← dimensione intervallo i-th simbolo della sequenza

return converter $\left(l_n + \frac{s_n}{2}\right)$ emit $\lceil \log_2 \frac{2}{s_n} \rceil$ bits

Truncating:

let's assume $d = \lceil \log_2 \frac{2}{s_n} \rceil \xrightarrow{\text{hence}} \leq 2^{-d} = \frac{1}{2^{\lceil \log_2 \frac{2}{s_n} \rceil}} \leq \frac{1}{2^{\log_2 \frac{2}{s_n}}} = \frac{1}{2/s_n} = \frac{s_n}{2}$

$x = \frac{13}{64} + \frac{1}{128} = \frac{39}{128}$ number to send for compression

$x = l_n + \frac{s_n}{2}$

$d = \lceil \log_2 \frac{2}{1/64} \rceil = \lceil \log_2 128 \rceil = 7$ → 0100111 bits to encode $\hat{x}$

Th. the number of bits emitted by aritmetic is at most $2 + n \cdot H$

$$\lceil \log_2 \frac{2}{s_n} \rceil = \lceil 1 - \log_2 s_n \rceil < 2 - \log_2 s_n = 2 - \log_2 \prod_{i=1}^{n} p(T[i]) =$$

$$= 2 - \sum_{i=1}^{n} \log_2 p(T[i]) = 2 - n \sum_{\sigma} \frac{\#\text{occ}(\sigma)}{n} \cdot \log_2 p(\sigma) = 2 + n \sum_{\sigma} p(\sigma) \log_2 \frac{1}{p(\sigma)} =$$

$$= 2 + n \sum_{\sigma} p(\sigma) \log_2 \frac{1}{p(\sigma)} = n \cdot H + 2 \text{ bits} \quad \blacksquare$$

bits per symbol $\frac{n \cdot H + 2}{n} = H + \frac{2}{n}$

Huff. compressed len $\leq n(H+1) = nH + n$ bits

Arith. compressed len $\leq nH + 2$ bits

Huff per symbol $\leq H + 1$

Arith. per symbol $\leq H + \frac{2}{n}$

Lempel-Ziv '77 '78 (dictionary based compression) → gzip, 7z, Brotly

- dictionary (77 → all substring, 78 → some substring) (hash table, trie) { <77, <778

- updating procedure for the dictionary

transform the text { parsing strategy

T → < d, l, char+1 >

T = A A C A A C A B C A A A | A A A A C

<6,3,A> <1,4,C>

dictionary = set of all substring starting before

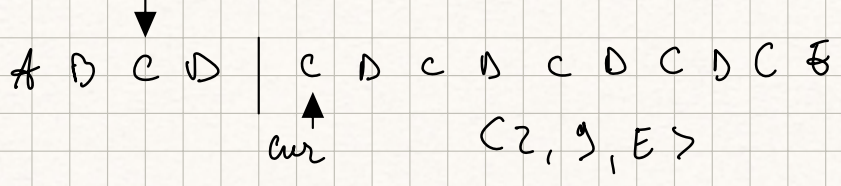
string for d $S_d = 0\ 1\ 3\ 6\ 4$

string for l $S_l = 0\ 1\ 4\ 3\ 4$

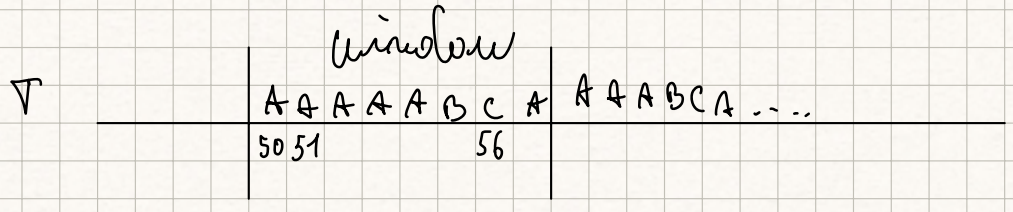
string for char $S_c = A\ C\ B\ A\ C$

} Huffman / Arithmetic

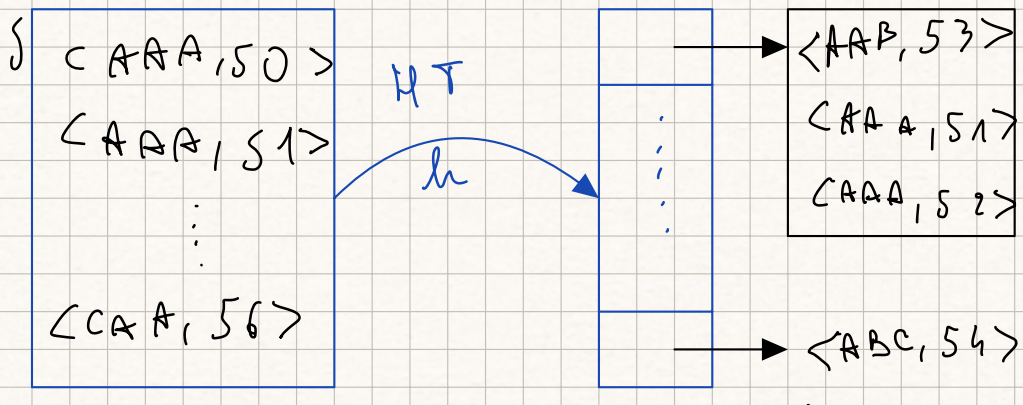
cur - d




```
for (i=0, i < len, i++)
    out[cur+i] = out[cur-off+i]
```



larger w → better compression

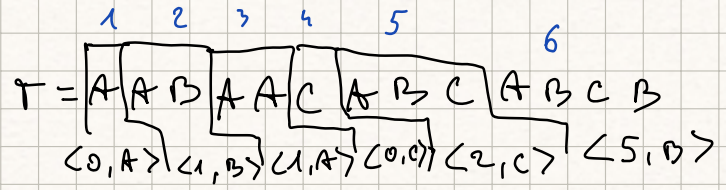


gzip implement 9 configuration that change the window of compression

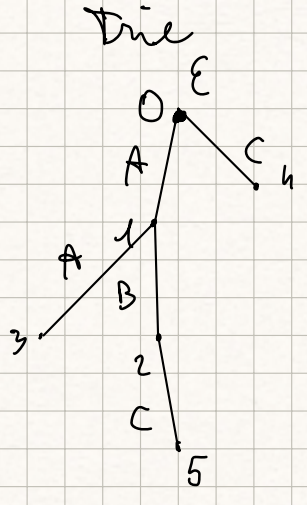
window = 1 → poor compression ratio

window = 9 → better compression ratio

LZ78 : we construct a trie data structure



- find the longest copy in the dictionary $\equiv S$
- emit ID of the longest copy founded and next char (pair)
- update the dictionary, adding S_c with new ID



Text formed by A^n char: LZ78 / LZ78

(2) <0,0,A>; <1, n-1, EOF>

$\sqrt{n}$

$1+2+3+\dots+V_i = n$

$\frac{V_i(V_i+1)}{2} = n$

Brotli, ZSTD

Canonical Huffman

ANS Asymetric Numeral System

b-zip

L = ipssmpissii

↑ 6

↳ < ipssmpissii, 6 >
L' preamble

Transforming local homogeneity into global homogeneity
① BWT
② move to front
③ Run length encoding (RLE∅)
④ Statistical coder

move-to-front coding (symbols → numbers)
transforms

input: (string, list of symbols)

wheeler's code

$\begin{cases} L' = ipssmpissii \\ \Sigma = \{1, 2, 3, 4\} \\ \{i, m, p, s\} \end{cases}$ → 1 3 4 1 4 3 4 4 1 2 1 11

preamble sip m 10 100

≥ 2 at least two bit
drop the first bit

symbols → RLE∅ 0 3 4 0 4 3 4 4 0 2 00

0 → 10 → 2 → 1
10 → 2 → 1
10 → 2 → 1
100 → 4 → 3

$\Sigma = \{0, 1, 2, 3, 4\}$
$P = \frac{4}{12}\ \frac{2}{12}\ \frac{1}{12}\ \frac{2}{12}\ \frac{4}{12}$

then apply Huffmann / Arithmetic