15/10

Typical $h: U \longrightarrow \{0, 1, \dots, m-1\}$ is called simple uniform if, given $k \in U$, $h(k) \to 0 \dots m-1$

most famous $h(k) = (k \bmod \underset{\text{random}}{p}) \bmod \underset{\text{fixed}}{m}$

$p \geq m$

The hash function belongs to a family $\mathcal{H}$

How big is $\mathcal{H}$ in order to guarantee that each key is in a different slot?

fixed k $\quad k \in U \to \left.\begin{matrix} 0 \\ 1 \\ \dots \\ m-1 \end{matrix}\right\}$ m possibilities $\forall k$

$k_1 \to m$
$k_2 \to m$
$\dots$
$k_U \to m$
$= m^U = |\mathcal{H}|$

so $|\mathcal{H}| = m^U$ so that it can represent any $f \in \mathcal{H}$

$\Rightarrow$ mod $\log_2 m^U$ bits $= U \log_2 m$ bits

if integers 64 bits, $U \to 2^{64}$

Impossible! We need to find a different approach $\equiv$ relax the property of "simple uniform"

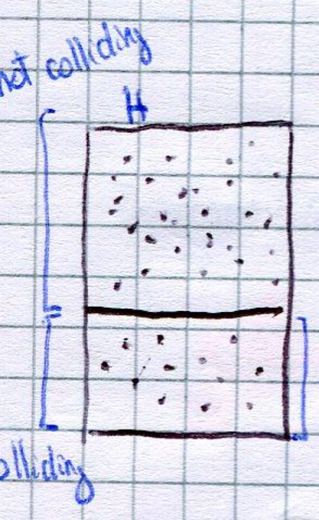
→ UNIVERSAL HASHING

Let H a set of functions $h \in H$, $h: U \to \{0, 1, \dots, m-1\}$

H is said to be UNIVERSAL if and only if (iff) $\forall x, y \in U$ it is

$$\underbrace{|\{h \in H . h(x) = h(y)\}|}_{\text{cardinality}} \leq \frac{|H|}{m}$$

#functions that collide



$\Rightarrow$ prob. that I get one that collide $\to \frac{1}{m}$

NOTE: c-UNIVERSAL $\to \frac{c \cdot |H|}{m}$ ← 1-universal

Theo Let $T[0, m-1]$ a table with chaining and pick $h \in H$ at random, then the expected length of a chain for the keys in a set S is $1 + \alpha$

$|S| = n$ $\quad \alpha = \frac{n}{m}$

DIM

$$I_{xy} = \begin{cases} 1 & \text{if } h(x) = h(y) \\ 0 & \text{o/w} \end{cases}$$

Given h, $\mathbb{P}(I_{xy} = 1) = \mathbb{P}(h(x) = h(y)) \leq \frac{1}{m}$

← def. of universality

$$\mathbb{E}[I_{xy}] = 1 \cdot \mathbb{P}(I_{xy} = 1) + \cancel{0 \cdot \mathbb{P}(I_{xy} = 0)} \leq \frac{1}{m}$$

avg length of a list

$$\underset{x}{1} + \sum_{\substack{y \in S \\ y \neq x}} I_{xy}$$

← random, so get the avg

$$\mathbb{E}\left[1 + \sum I_{xy}\right] = 1 + \sum_{\substack{x \neq y \\ y \in S}} \mathbb{E}[I_{xy}]$$

$$\leq 1 + \frac{n-1}{m} < 1 + \frac{n}{m} = 1 + \alpha$$

Provided that h exists!

Theo $\alpha = \Theta(1)$ α is constant

if $m = \Theta(n) \Rightarrow$ avg list length is a constant $\Theta(1)$ AND expected length of the longest list $O\left(\frac{\log n}{\log\log n}\right)$

DIM

$Q(k)$ = prob. that a given list contains k items

$$\leq \underbrace{\binom{n}{k}}_{\text{all the ways to get the keys}} * \underbrace{\left(\frac{1}{m}\right)^k}_{\text{that slot (k times)}} * \underbrace{\left(1-\frac{1}{m}\right)^{n-k}}_{\text{not that slot}} < \left(\frac{e}{k}\right)^k$$

← Stirling's approximation

NOTE $k \geq 3 \rightarrow \frac{e}{k} < 1 \Rightarrow \left(\frac{e}{k}\right)^k$ is decreasing $< \frac{1}{n^3}$

$$\mathbb{P}(\overset{\text{longest list}}{H} = k) = \mathbb{P}(\overset{\text{slot}}{\exists i} : \overset{\text{\#items that slot is k}}{N(i) = k} \wedge N(s) \leq k \quad \forall s \neq i)$$

$$\leq \mathbb{P}(\exists i : N(i) = k) \leq m \cdot Q(k)$$

[def] $$\mathbb{E}[H] = \sum_{k=0}^{n} k \cdot \mathbb{P}(H=k) = \underbrace{\sum_{k=0}^{k_0} k \cdot \mathbb{P}(H=k)}_{\leq k_0 \sum_{k=0}^{k_0} \mathbb{P}(H=k) \;=\; k_0 \cdot \mathbb{P}(H \leq k_0) \;\leq\; k_0} + \underbrace{\sum_{k=k_0+1}^{n} k \cdot \mathbb{P}(H=k)}_{\leq n \cdot \sum_{k=k_0+1}^{n} \mathbb{P}(H=k) \;=\; n^2 \cdot \sum_{k=k_0+1}^{n} \cancel{n} \cdot Q(k) \;\leq\; \cancel{n^2} \cdot \sum_{k=k_0+1}^{n} \frac{1}{n^{\cancel{3}}} = \sum_{k=k_0+1}^{n} \frac{1}{n} \leq \sum_{k=1}^{n} \frac{1}{n} = 1}$$

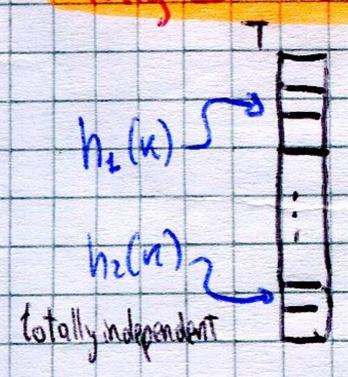
sum of probs

$\mathbb{P}(\cdot) \leq 1$

$$\Rightarrow \mathbb{E}[H] \leq \underbrace{k_0}_{\frac{\log n}{\log\log n}} + 1$$

STRONG RESULT hash tables with chaining are very strong

"Why do we use just one hash function?" – Harvard guy



chose always the one ↑ (h_1)

Avg length of the longest list is $O(\log\log n)$

using d hash functions? ↑ is $O\left(\frac{\log\log n}{\log d}\right)$

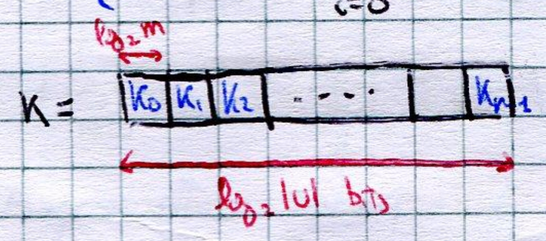
$d=2 \Rightarrow 1$ ✓ 2 best trade off

$d>2$ no real advantages

H_a = H parametized by $0 \leq a < |U|$

$$H_a = \left\{ h_a(k) = \sum_{i=0}^{r-1} a_i \cdot k_i \mod m \right\}$$ prime

$k \in U$ k consists of $\log_2 |U|$ bits



partition the bit streams into blocks of $\log_2 m$ bit

(m size of table) $\Rightarrow$ #blocks $= \frac{\log_2 |U|}{\log_2 m} = r$

a = integer of size $\log U$ $\boxed{a_0 \mid a_1 \mid \dots \mid a_{r-1}}$

to generate a random hash function, I generate a

How many as I have? 000...00 → 111...11 $\equiv 0 \rightarrow U-1$

$$\cancel{\log_2} |U| = r \log_2 m = \cancel{\log_2} m^r \Rightarrow |U| = m^r$$

→

← Is it an universal class?

Fixed x, y ($x \neq y$) count # hash functions $h_a(x) = h_a(y)$

$$\#a : \sum_{i=0}^{r-1} a_i x_i = \sum_{i=0}^{r-1} a_i y_i \bmod m$$

Since $x \neq y$, they have to differ on at least one position, lets assume its $[0]$ → x_0, y_0

$$\Rightarrow \sum_{i=0}^{r-1} a_i (x_i - y_i) = 0 \bmod m$$

Take off block #0 $\Rightarrow a_0 \underbrace{(x_0 - y_0)}_{\neq 0} + \sum_{i=1}^{r-1} a_i (x_i - y_i) = 0 \bmod m$

$\Rightarrow \exists (x_0 - y_0)^{-1}$ "exists the inverse" $[(x_0-y_0)^{-1} \cdot (x_0 - y_0)]$

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

$$a_0 = -(x_0 - y_0)^{-1} \underbrace{\sum_{i=1}^{r-1} a_i (x_i - y_i)}_{\text{choose } a_1, a_2, \dots, a_{r-1} \text{ as you want}} \bmod m$$

fixed, can't choose

$\Rightarrow$ all possible ways to choose $a_1, \dots, a_{r-1}$

$a_1 \to [0, m-1]$, $a_2 \to [0, m-1]$, ... $\to m^{r-1}$ ways to choose a

$$\frac{m^r}{m} \longleftrightarrow |U| = \frac{|U|}{m} = \frac{H}{m}$$ for every number $0 \dots n-1$ I have an hash function ≡ def. of universal

$\Rightarrow$ UNIVERSAL HASH FUNCTION EXISTS! ☺

16/10

$$h_a(k) = (\underbrace{a * k}_{\text{fast}} \underbrace{\bmod 2^h}_{h \text{ bits}}) \text{ div } 2^{h-\ell}$$

$a*k$

ℓ bits $\Rightarrow 0 < num < \overset{2^\ell}{\overset{|||}{m}}$ prob. collisions $\leq \frac{2}{m}$

So we have everything we needed, but still in the worst case we have to scan the list

lost the good thing (constant time access) about hashing with chaining!!

PERFECT HASHTABLE

An hash function h from $U \to \{0, \dots, m-1\}$ is perfect (no collisions) with respect to a dictionary (static, no add/...)

iff $\forall k, k' \in S$, $h(k) \neq h(k')$

fixed respect to S, change $S \Rightarrow$ change h

of course, $|S| = n$ and $n \leq m$

How to build h, given S?

if $m = n$ MINIMAL perfect hash function (table is full)

Theo 1st part: if we store q keys and an hashtable of size $w=q^2$ using an universal hash function h the expected #collisions among the q keys is less than $\frac{1}{2}$.

2nd part: Consequently, the probability to have a collision is $< \frac{1}{2}$.

1st part tells us that there are spots with 0 collisions $\Rightarrow$ it's possible to design perfect hashtables!
2nd part tells us we can construct it ($\nearrow$) efficiently.

$\rightarrow$ hash function that produces $\lg_2 q^2$ bits
$2\lg_2 q$

DIM

$$q \text{ keys} \Rightarrow \underset{\substack{\text{max num of collisions}}}{\text{tot. num. of pairs}} = \binom{q}{2} = \frac{q\cdot(q-1)}{2} < \frac{q^2}{2}$$

Picked an universal hash function $h: U \rightarrow w = q^2 \equiv \mathbb{P}(2 \text{ collide}) \leq \frac{1}{q^2} = \frac{1}{w}$
given 2 keys, $\mathbb{P}(\text{collision}) = \frac{1}{m}$

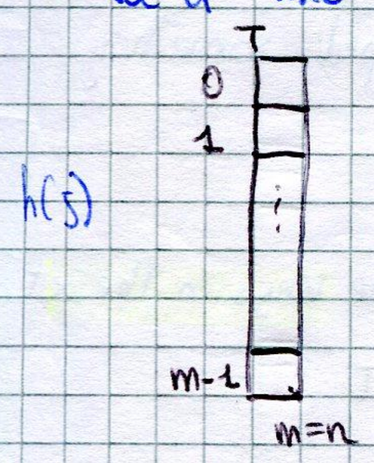
$$\mathbb{E}[\#\text{collisions}] < \frac{\cancel{q^2}}{2} \cdot \frac{1}{\cancel{q^2}} = \frac{1}{2}$$ 1st part ✓

#collisions $\in \mathbb{N}$, if $\mathbb{E} = \frac{1}{2} \Rightarrow \exists$ "0 collisions" for sure $\Rightarrow \mathbb{P}(\geq 1) = ?$

Markov Bound $\mathbb{P}(\underset{\#\text{collisions}}{X} \geq \underset{2}{t} \cdot \underset{\frac{1}{2}}{\mathbb{E}[X]}) \leq \frac{1}{t}$

$\mathbb{P}(\#\text{collisions} \geq 1) \leq \frac{1}{2} \Rightarrow$ a collision $\equiv \mathbb{P}(\text{fail to have a perfect hash})$

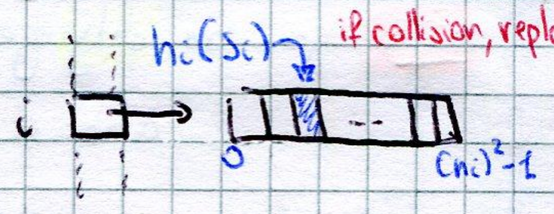
we'd like something $\Theta(\#\text{keys})$



$n_0 = \#\text{keys}$ ending in slot 0
$n_1 = \#\text{keys}$ " 1
s_i

Idea if $\sum_{i=0}^{m-1} (n_i)^2 < 2n$ PROCEED

else pick another $h(i)$



$\forall$ slot $\Rightarrow$ m hash functions (1 per slot)

if true, size of ALL tables $< 3n$ (linear)

$\Rightarrow$ given $x \in S$ $\quad h(s) \rightarrow i \rightarrow h_i(s) \rightarrow$ check

If $m=n$, having h univ. hash func. ensures that $\mathbb{E}[\sum_i (n_i)^2] < 2n$

$$\mathbb{E}\left[\sum_{i=0}^{m-1}(n_i)^2\right] = \mathbb{E}\left[\sum_{i=0}^{m-1} n_i + 2\binom{n_i}{2}\right] = \underbrace{\mathbb{E}\left[\sum_{i=0}^{m-1} n_i\right]}_{n} + 2\,\mathbb{E}\left[\sum_{i=0}^{m-1}\binom{n_i}{2}\right]$$

$a^2 = a + 2\binom{a}{2} = a + \cancel{2}\cdot\frac{a(a-1)}{\cancel{2}}$ ✓

constant to the random pick of h

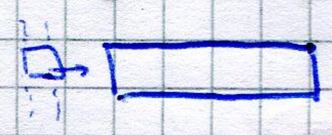
all the possible pairs I can form in a slot "#collisions in slot i"

#collisions on all table by H
$\equiv \binom{n}{2}\cdot\frac{1}{m} = n$
#pairs * prob collision
$\frac{n(n-1)}{2}\frac{1}{n} = \frac{n-1}{2}$

$$= n + \cancel{2}\left(\frac{n-1}{\cancel{2}}\right)$$

$$n + n - 1 < 2n$$

$\Rightarrow$ at some point, the if will be True $\Rightarrow$ ✓

in  it's ok because n_i keys in n_i^2 space, ok for previous lemma

at exam
Ist lvl: $a\,x\,(+b) \bmod n$
IInd lvl: $a\,x\,(+b) \bmod (n_i)^2$

exercise $S = \{98, 19, 14, 50, 1, 72, 79, 3, 69\}$ construct a perfect HT.

$h(x) = ((2x+42) \bmod 101) \bmod 11$ $n=9$

0	→ 1 → [1]
1	
2	→ 69 → [69]
3	→ 19, 98
4	→ 3, 14
5	
6	
7	
8	→ 50, 72
9	→ 79 → [79]
10	

sum of squares: $1+1+4+4+4+1=15$ $15 < 2*9 = 18$? ✓ PROCEED

$h_0(x) = x \bmod 1$ only 1 way in 1^2 spot

same goes for $h_2(x)$ and $h_9(x)$

In 3, 4 and 8 we take a table of size 4 (or next prime)

How to manage a size not prime? (like 4 ↗)

$(ax+b \bmod \underset{\text{prime}}{p}) \bmod (n_i)^2$ $p > n_i^2$

or just hope and try mod n_i^2 :)

$h_3(x) = (4x+42 \bmod 101) \bmod 4$

[3] →		19	98	
	0	1	2	3

$h_4(x) = (3x+42 \bmod 101) \bmod 4$

[4] →	14			3
	0	1	2	3

$h_8(x) = (4x+42 \bmod 101) \bmod 4$

[8] →	50			72
	0	1	2	3

we have to store the hash functions too, but each takes 3 components

21/10

MINIMAL ORDERED PERFECT HASHING

uses exactly n space — $\forall x, y \in S.\ x < y \Rightarrow h(x) < h(y)$ (LEXICOGRAPHIC NAMING) — no collisions for a set of keys S

combine

i-th string gets the ~~#~~ i-th-1 slot

1st gets 0

useful when you want to have mapped something in the table

NOTE you can only get the lexic. order for the keys in the

$abba \notin S,\ h(abba) = x \wedge x \in T$

① Pick 2 universal hash function $U \to \{0, 1, \dots, m'-1\}$ $m' = c \cdot m$ $[c \geq 1$ constant$]$

h_1, h_2 $ax+b \bmod m$ this?

② COMPUTE function $g: [m'] \to [n]$

$h(t) = g(h_1(t)) + g(h_2(t)) \bmod n$

$\leftarrow m' = 12$ $(12 > 9)$

	key	h_1	h_2
0	body	1	6
1	cat	7	2
2	dog	5	7
3	flower	4	6
4	house	1	10
5	mouse	0	1
6	sun	8	11
7	tree	11	9

$h(body) = g(1) + g(6) \bmod 9 = 0$

$h(cat) \to 1 = g(7) + g(2) \bmod 9$

$h(dog) \to 2 = g(5) + g(7) \bmod 9$

Assuming that g exists:

space $h_1 \wedge h_2$ take constant space (3 mts each)

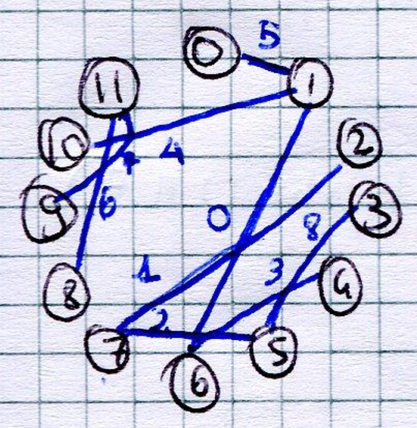
g is a table (tabular function) → store $m'-1$ edges $O(m')$ = $O(n)$

time $h_1 \wedge h_2$ take constant time to evaluate

g constant time $O(1)$

ALGORITHM ① **create a graph G** that consists of m' nodes and n edges

NOTE n used for edges, not for nodes, because it's the #keys



How to draw the edges?

edge ∀key t $(h_1(t), h_2(t))$

⇒ graph is **sparse** (#nodes > #edges)

undirected

random

We need an **Acyclic graph**

Since G is sparse ⇒ $\mathbb{P}$(G acyclic) is very high ≡ few exec of the alg is enough

(to check if it's cyclic it's linear, just visit)

if repeat: redraw $h_1 \wedge h_2$

② **start from a node (0) and assign a number** (0 xd)

⇒ every node corresponds to an equation

$(0,1) \xrightarrow{\text{"must return"}} 5$ $\quad 5 = \overset{0\ xd}{g(0)} + g(1) \mod g$ ⇒ $g(1) = 5$

⇒ $(1,6) \to 0$ $\quad 0 = \underset{5}{g(1)} + g(6) \mod g$ ⇒ $g(6) = 8$

...

If there was a cycle, at some point I'd got an unsolvable equation!

When you finish the components on one part, restart with another one (0 xd)

x	g(x)
0	0
1	5
2	0
3	7
4	8
5	1
6	4
7	1
8	6
9	7
10	8
11	0

cat? → $h_1(\overset{7}{cat}) + h_2(\overset{2}{cat})$

$g(7) + g(2)$

$0 + 1 = 1 \mod g = 1$ correct, cat is the 2nd key in S

key ∉ S? mom → $h_1(mom) + h_2(mom)$

ex 2 10

$g(2) + g(10)$

$0 + 8 \to 8$ 2 errors: not only mom ∉ S but even the spot is useless, "m" lexic. is in 4-5

jump + jump + 3rd cache miss to check

Cuckoo Hashing

We want:
- $O(1)$ worst case time for a query (here 2 mem access) (also deletion)
- $O(1)$ ammortized avg insertion

$h_1 \wedge h_2$ U.H.F.

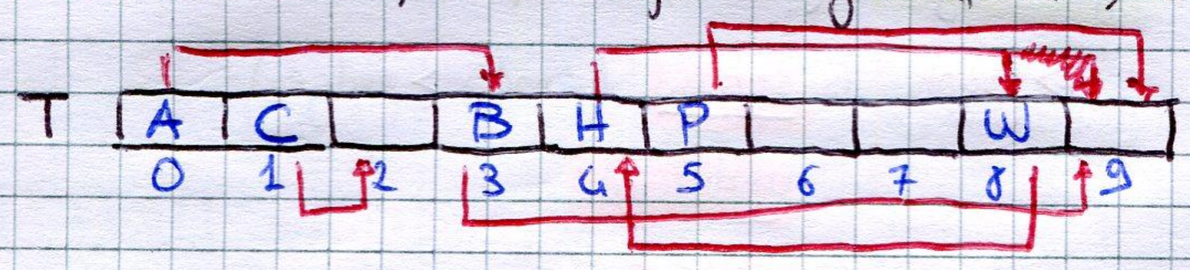
T_1 [ ] use h_1 to access T_1

T_2 [ ] use h_2 to access T_2

(or just one longer T and use both $h_1 \wedge h_2$)

$\forall key \in S$: $h_1(k), h_2(k)$ [2 positions]

$m = |T|$ = #nodes of G_T, n = #keys = #edges of G_T, G_T is random ($\Leftarrow h_1 \wedge h_2$ U.H.F.)



every cell is a node of a graph G_T

every key induces a random edge (directed) $(h_1(k), h_2(k))$ $h_1(A)=0$ $h_2(A)=3$

$\equiv$ pointer says "A might be here or there"

NOTE in the ex, H and W are INTERCHANGEABLE

Insertion

- at least one spot free ex D, $h_1(D)=0$ (occupied) $h_2(D)=2$ (free) $\rightarrow$ D stored in 2 AND put an edge from 2 to 2
- both occupied Cuckoo $\equiv$ kicks another one from the nest

  ex $h_1(D)=0$, $h_2(D)=3$

  start from $h_1 \rightarrow 0$ includes A

  D stored in 0, kicks A

  A follows the edge $\rightarrow$ B (3)

  B kicked, goes to 9 (free) ✓

  when this happens $\Rightarrow$ change direction of the pointers!

  $A \rightarrow B \rightarrow \lrcorner \Rightarrow D \leftarrow A \leftarrow B$

- What if "cycles"?

  $h_1(F)=4$ when h_1 fails, try h_2

  $h_2(F)=1$

- if both cycle?? $h_1(F)=4$ $h_2(F)=\{0,4,3,1\}$

  [implementing note max n steps, after n steps $\wedge$!stopped $\Rightarrow$ cycle]

  can't

$\Rightarrow$ Prob. of no possible insertion? $\Rightarrow$ $\exists$ 2 cycles (minimum condition) $<$ $\exists$ 1 cycle (upperbound) $\Rightarrow$

Theo

$\forall$ pair of entries i, j consider the path $i \xrightarrow{l} j$ with length l the shortest path from i

If $m \geq \underbrace{2 \cdot c \cdot n}_{\text{very sparse}}$ (#nodes) $\quad \mathbb{P}(\text{shortest path} = l) \leq \frac{c^{-l}}{m} = \frac{1}{m \cdot c^{l}} \qquad c > 1$

$\Rightarrow$
- longer path $\equiv$ smaller prob
- reduction is exponential in c

DIM

Base $\boxed{l = 1}$

$P(i \overset{l}{\rightsquigarrow} j)$

$\downarrow$

$P(i \longrightarrow j)$

(upperbound by undirected)

$\longrightarrow$

$\exists k.\ h_1(k) = i \wedge h_2(k) = j \quad \vee$

$\vee \quad h_1(k) = j \wedge h_2(k) = i$

$\longleftarrow$

$h_1 \wedge h_2$ univ. $\Rightarrow \mathbb{P} \leq \frac{1}{m}$ [size of Table]

$\Rightarrow \quad \uparrow \leq \frac{1}{m^2} \vee \frac{1}{m^2}$

$\equiv \mathbb{P}(i - j) \leq \frac{2}{m^2}$ prob. for a specific key

n keys $\longrightarrow \frac{2n}{m^2} \qquad m \geq 2 \cdot c \cdot n \rightarrow \frac{\cancel{2}}{m^{\cancel{2}}} \cdot \frac{\cancel{m}}{\cancel{2}c} = \frac{1}{m \cdot c}$ ✓

Ind $\boxed{\text{True for } l-1}$

$\mathbb{P}(i \xrightarrow{l-1} j) \leq \frac{1}{m \cdot c^{l-1}}$

up.bound m * prop fixed instead alone

$\mathbb{P}(i \xrightarrow{l} j) = \mathbb{P}(i \xrightarrow{l-1} h \xrightarrow{1} j) \leq m \cdot \mathbb{P}(\text{fixed } h \ \underset{|||}{i \xrightarrow{l-1} h} \xrightarrow{1} j)$

$\uparrow$ 7h line this

$m \cdot \frac{1}{m \cdot c^{l-1}} \cdot \frac{1}{m \cdot c}$

$= \frac{1}{m \cdot c^{l}}$

$\mathbb{P}(\text{collision}) = \mathbb{P}(\exists l : x \xrightarrow{l} y)$

$\leq \sum_{l=1}^{m} \frac{1}{m \cdot c^{l}} < \sum_{l=1}^{\infty} \frac{1}{m \cdot c^{l}} \simeq O\left(\frac{1}{m}\right)$

$\sum_{l=1}^{\infty} c^{-l} = \frac{1}{c-1}$

22/10

Yesterday we proved that $\mathbb{P}(i \xrightarrow{L} j) \leq \frac{1}{m \cdot c^L}$ if $m \geq 2 \cdot c \cdot n$

$\Downarrow$

table size is twice the needed (at least)

$\equiv$

occupancy < 50% = tons of wasted space!

$$\mathbb{P}(\exists i : i \circlearrowleft) \leq m \cdot \mathbb{P}(\text{specific } \hat{i} \text{ has a cycle})$$

(cycle)

$$= m \cdot \mathbb{P}(\exists L : h \xrightarrow{L} h)$$

$$\leq m \cdot \sum_{L=1}^{n} \mathbb{P}(\text{path of a specific length } L : h \xrightarrow{L} h)$$

$$< \cancel{m} \cdot \sum_{L=1}^{\infty} \frac{1}{\cancel{m} \cdot c^L} = \frac{1}{c-1}$$

fix $c=3$ $(m \geq 6n)$

$\mathbb{P}(\exists \text{ cycle}) \leq \frac{1}{2}$

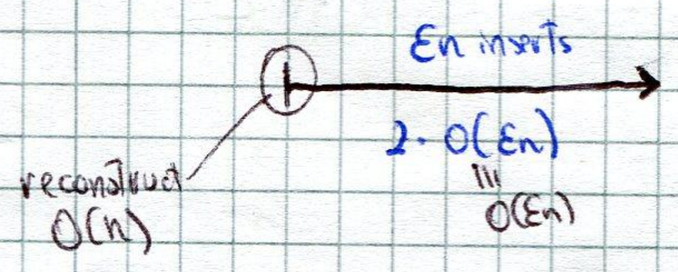
① We create a Cuckoo HashTable able to allocate $(1+\varepsilon)^* n$ keys by assuming that n is the current number of keys

② We create a table of size $m = 2^* c^* (1+\varepsilon)^* n$ (to apply the theo) $\Rightarrow$ up to εn insertions will have the hyp from the theo true

$\Rightarrow$ prob(fail insertions) $\leq$ prob($\exists$ cycle) $\leq \frac{1}{2}$

every insert has a prob. to fail $< \frac{1}{2}$

$\Rightarrow$ Facts
- $m = 2^* c^* (1+\varepsilon)^* n$
- $\mathbb{P}(\text{fail}) < \frac{1}{2}$ (2 reconstructions on average)
- Executed εn insertions
- avg length of a chain is constant $\Rightarrow$ avg cost of an insertion is $O(1)$

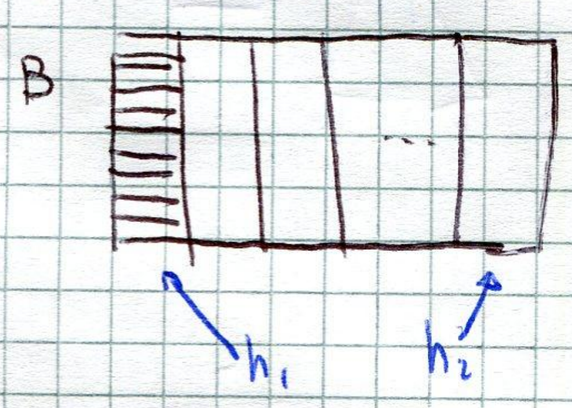


$$\equiv \underbrace{O(n + \varepsilon n)}_{\varepsilon n} = O(1) \text{ avg ammortized}$$

#op. needed

In practice

Cuckoo, instead of table of slots, its a table of buckets



$\log \log n$ on avg length (#items)

COMBINING CUCKOO $\wedge$ 2 LEFT HASHING

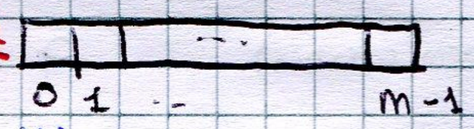
$\Rightarrow$ 95% occupancy

fix B to 5 $\rightarrow$ very rare

if more than 5, rehash one

Last kind of hashtable we discuss: **Bloom Filter** one-side error data structure = data struct that can do error
bounded in prob.

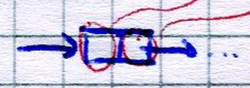
Idea drop keys to save space

① **Bit array** $B =$ 

$B[i] = \{0,1\}\ \forall i$

Note in Cuckoo $m \cdot \lg_2 U$ bits (key length)

in Hash Chain $m \cdot \lg_2 m + n(\lg_2 U + \lg_2 n)$ (#objects)



② **Set of U.H.F.** $\{h_1, h_2, \dots, h_k\}$

$h_i : U \to \{0, \dots, m-1\}$

How to **insert** a key?

Insert (key) $\longrightarrow$ COMPUTE $B[h_i(key)] = 1$

if $k=3 \to$ I set three bits to 1, others are 0s

cost $O(k)$ time

Note: Delete (key) not supported

Query (key)? $k \in$?

- if $\exists i : B[h_i(key)] = 0 \Rightarrow \notin$ for sure if key $\in$ set $\Rightarrow$ those bits would be all 1
- else ¿(ツ)ノ the bits could have been set by other keys!

$\searrow$ return yes, it $\in$ set, but might be an error

prob.?

$P(\text{fixed } x : B[x] = 0)$ if size $|B| = m \wedge$ inserted n keys

$\forall$ key, set k positions $\to$ AT MOST $n*k$ bits set to 1

$= \left(\frac{m-1}{m}\right)^{n*k}$

m slots, 1 prohibited, for each key

$$= \left(1 - \frac{1}{m}\right)^{\frac{n \cdot k \cdot m}{m}} \qquad \left[\underbrace{\left(1 - \frac{1}{m}\right)^{m}}_{e^{-1}}\right]^{\frac{nk}{m}} \longrightarrow e^{-\frac{nk}{m}}$$

$\Rightarrow$ **P(Query makes an error)** $= P(\text{given } k \notin D, B[h_i[k]] = 1) \simeq \left(1 - e^{-\frac{nk}{m}}\right)^{k}$ (k bits are 1)

the error decreases
- increasing the size of B (more space)
- increasing the #hash functions (more time)

minimum error "when there are two variables, fix one and optimize the other one"

Fix m, minimize error with respect to k $\quad k_{opt} = \frac{m}{n} \ln 2 \longrightarrow \varepsilon_{opt} \simeq (0.685)^{m/n}$

this way, how many bits are 0s and 1s?

substitute k_{opt} in $e^{-\frac{nk}{m}}$

$$e^{-\frac{m}{n} \frac{\ln 2 \cdot n}{m}} = \frac{1}{e^{\ln 2}} = \frac{1}{2}$$ Optimal choice

half 0s, half 1s

OBV if #0s > #1s $\to$ wasting space

if #1s > #0s $\to$ increasing errors

NOTE if $m = c \cdot n$ $\Rightarrow$ increase size table (c)

#functions increases LINEARLY #errors decreases EXPONENTIALLY

ex $m = 30n$

$k_{opt} = 30 \cdot \ln 2 \approx 45$ hash functions (a.k.a. mem. accesses) 5 is ok to not overkill it

ex $c = 30$ times #keys $\rightarrow 30n \rightarrow$ space is $30n$ bits $\approx 4n$ bytes

$\forall$ key I'm spending 4 bytes

error $= (0.685)^{30} \rightarrow$ LOW

Better than previous when length(key) + pointer > 4B

it's 4B...

so don't use only if strings are very short!

4/11

More data structures... Randomized Data Structures → TREAP / SKIP-LIST

TREAP BinTree over keys $\wedge$ (min) Heap over priorities $\Rightarrow$ items are <key, priority>

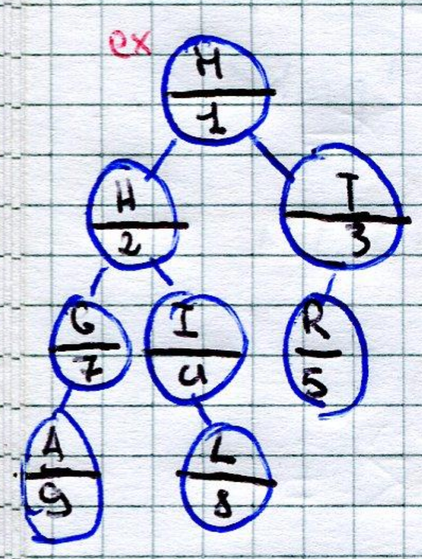
Tree + Heap

FOR SIMPLICITY <LETTERS, NUMBERS>

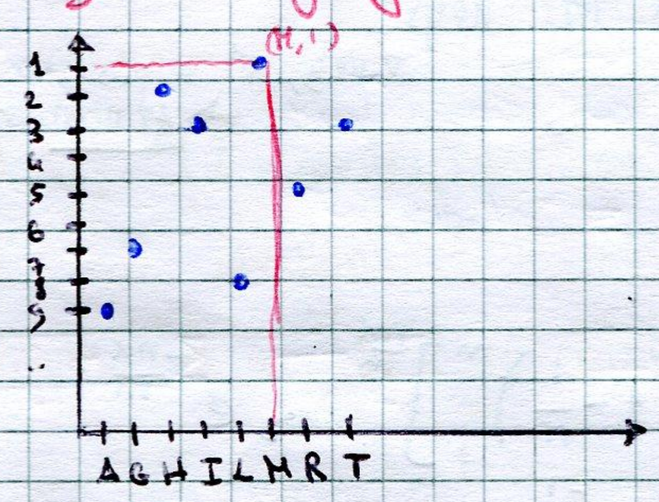
$\Rightarrow$ It satisfies 2 properties ① it's a BST over keys

② it's a Heap over the priorities

ex



very interesting geometric interpretation



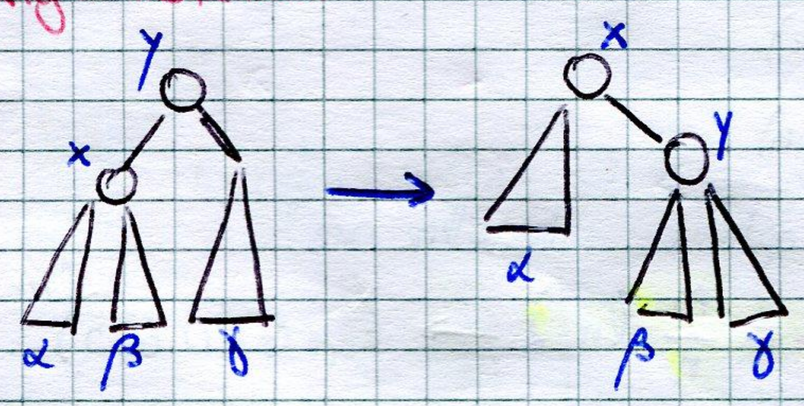
Pick (H,1) $\equiv$ divide in 2 parts

left Treap $\wedge$ right Treap

then pick (H,2),... and continue dividing into subquad

Rotations

Right Rotation



preserved $\alpha < x < \beta < y < \gamma$

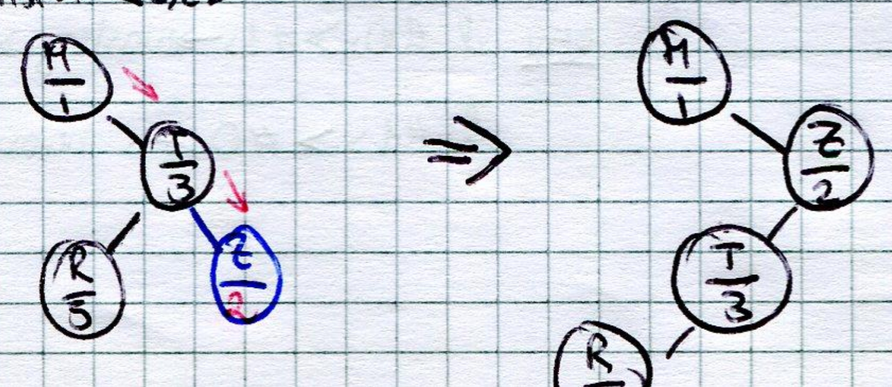
Left is simmetric

Insert

① Insert the key as in the BST $\equiv$ search in the Tree, when you go out of the Tree insert

② Insert the priority and fix the Heap with rotations

ex insert <Z,2>



cost (rotation) constant, change 2 pointers $O(1)$

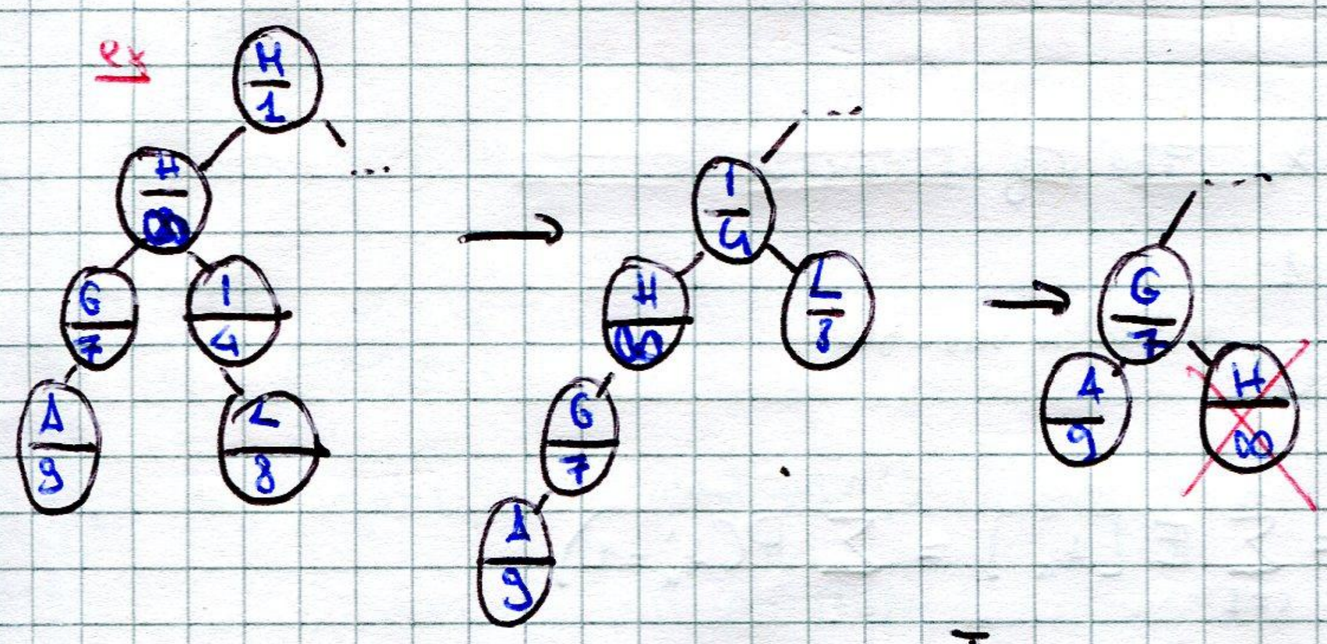
cost (insert) depth of tree ↓ + rotation ↑ $O(\text{depth})$

cost $O(\text{depth})$

Deletion

- 0 or 1 children → easy
- 2 childrens: ⇒ transform the node in a leaf ≡ set priority to $+\infty$

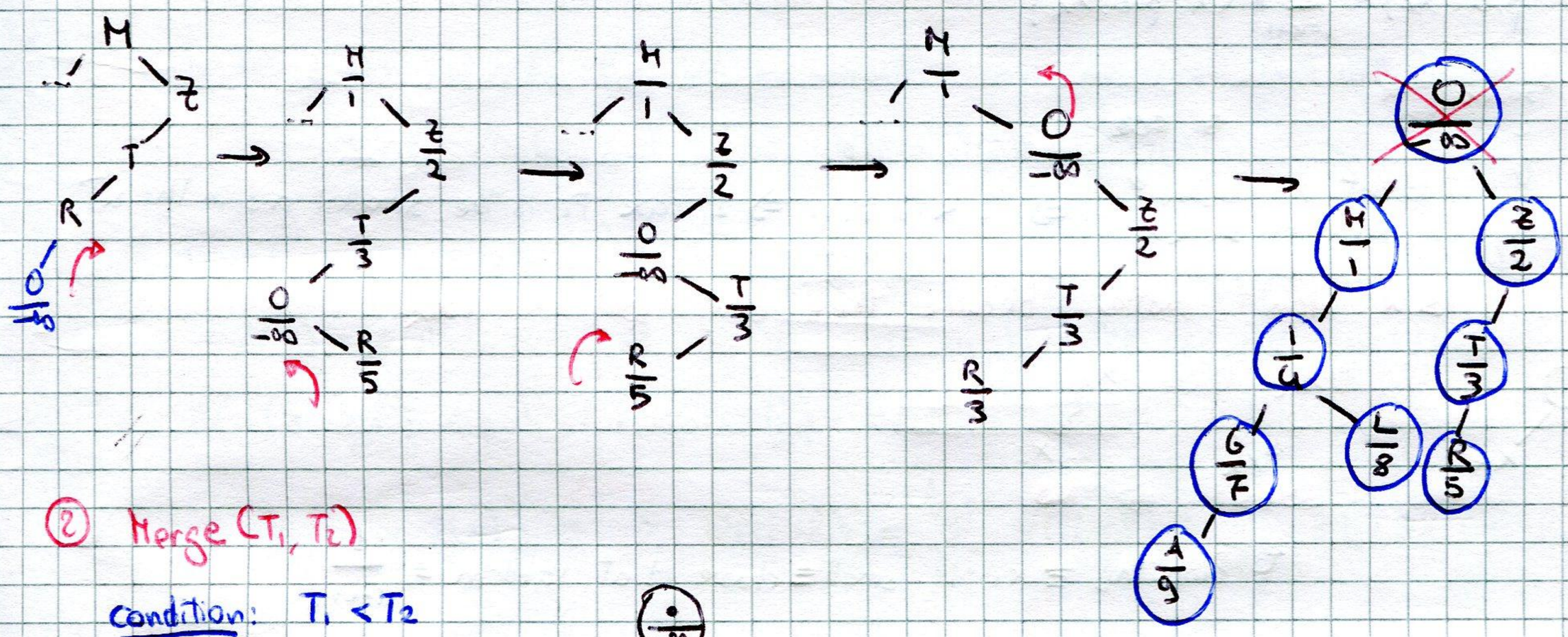
ex



2 other operations: ① split(k) < $T_{\le k}$, $T_{>k}$ generate 2 Treaps

split(O) ≡ create a treap with root ≡ <O, −∞>, then remove it

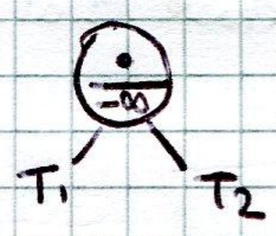
cost insert :)



② Merge(T_1, T_2)

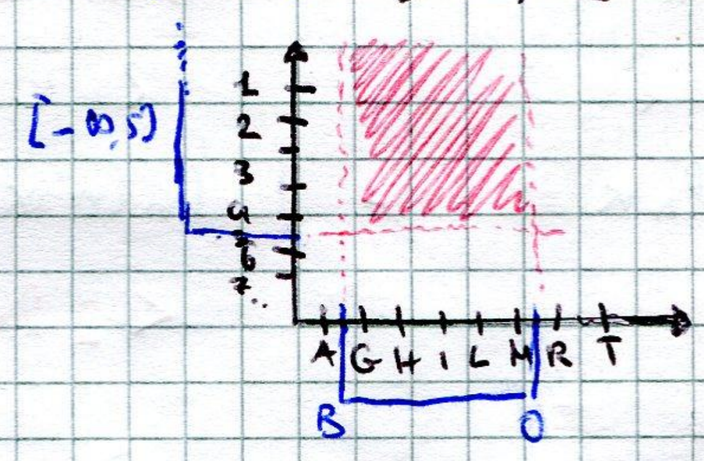
Condition: $T_1 < T_2$

put a dummy root



This data structure can solve 3-sided range query ≡ Query with < x-range [B, O] ∧ y-range [−∞, 5]

with a binTree you'd get all the points between [B, O] and then scan [−∞, 5] to drop.



How to "[B, O]"?

split("O") → $T_{\le O}$, $T_{>O}$ cost O(depth)

split("A", $T_{\le O}$) → $T_{\le A}$, $T_{>A, \le O}$
key before B
≡ $T_{\ge B, \le O}$

when checking a prio > the one I need → STOP
else return it and recurse

NOTE if it's [5, +∞] it's a Max-Heap!

How To balance it?

Random Priorities + TREAP

Given Dictionary D of keys, draw random priorities and build the TREAP

$\Rightarrow$ Balanced BST

Dim

sort D ideally $D = x_1 \; x_2 \; x_3 \; \dots \; x_n$ $\qquad x_k$ the key of rank k

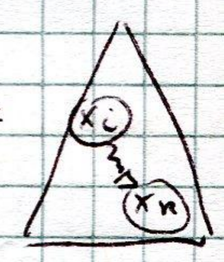
rank $r_1 < r_2 < r_3 \dots < r_n$

$A_k^i = [x_i$ is a proper ancestor of $x_k] = \begin{cases} 1 \\ 0 \end{cases}$ indicator variable

the event such That...

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

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



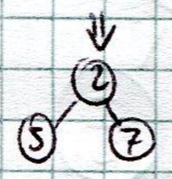
2 situations: ① $x_i < x_k$ ② $x_k < x_i$ $\leftrightarrow$ simmetric, consider only one

$$\sum_{i=1}^{k-1} \mathbb{P}(x_i \text{ p.a. } x_k) + \sum_{i=k+1}^{n} \mathbb{P}(x_i \text{ p.a. } x_k)$$

$x_i \quad x_{>i,<k} \quad x_k$

5 $\quad$ 2 $\quad$ 7

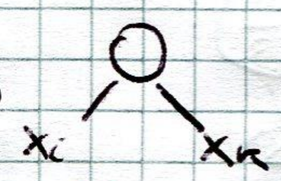
$\Downarrow$



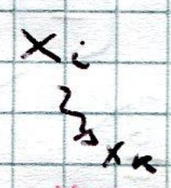
so not x_k for sure * not elems $>i$ $<k$ ... $\Rightarrow P_i < P_{\text{all others}} \Rightarrow$ 1 case, P_i is the smallest prio in The

$\mathbb{P}(x_i)$

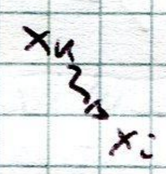
elems $<i$ and $>k$ don't matter because they don't split x_i & x_k

$\Rightarrow$ 

NO



YES



NO

Prob. this $\equiv x_i$ is small $\equiv$ choose 1 at random $= \frac{1}{k-i+1}$

in the other one it's $\frac{1}{i-k+1}$

$$= \sum_{i=1}^{k-1} \frac{1}{k-i+1} + \sum_{i=k+1}^{n} \frac{1}{i-k+1}$$

$\frac{1}{k} + \frac{1}{k-1} + \frac{1}{k-2} + \dots + \frac{1}{2} \qquad \frac{1}{2} + \dots + \frac{1}{n-k+1}$

$H_k - 1 \quad + \quad H_{n-k+1} - 1$

sum armonic numbers $\quad$ missing +1

$\ln k \quad + \quad \ln(n-k+1) \quad -2 \quad \equiv$ depends on k

$< 2 \ln n \; [-2]$

$\Rightarrow$ avg depth$(x_k) \overset{(2)}{<} \ln n$

5/11

SKIP-LIST $\boxed{2} \to \boxed{5} \to \boxed{6} \to \boxed{10} \to \boxed{15} \to \boxed{20}$ comparison-based data structure (DYNAMIC)

for each item toss a coin $\{H, T\}$ $\mathbb{P}(H) = \mathbb{P}(T) = \frac{1}{2}$

Idea

$n/2 \to$ H $\boxed{2}$ T T $\boxed{6}$ H $\boxed{10}$ H T T

$n/4 \to$ $\boxed{6}$ H $\boxed{10}$ H

$n/8 \to$ $\boxed{6}$ T

(... log #item)

#items

entrance

$\Rightarrow$

$-\infty \to 6 \to +\infty$

$-\infty \to 6 \to 10 \to +\infty$

$-\infty \to 2 \to 6 \to 10 \to +\infty$

$-\infty \to 2 \to 5 \to 6 \to 10 \to 15 \to 20 \to +\infty$

anticipate ← A DUMMY NODE → posticipate

$\Rightarrow$ Height is a constant (2 items)

focusing on an item

$Q(16)$ $-\infty \to \boxed{6} \to \boxed{10} \to \boxed{10} \to \boxed{15} \to \boxed{20}$

16 $\notin$ List

cost #lvls * # "ToTheRight"

NOTE In the INSERT the node that led me somewhere are the ones that will change!

$\Rightarrow$ Ins/Del cost: searchTime + $\underbrace{\#levels}_{\text{constant on avg}} * O(1)$

① create logarithmic #levels (w.h.p.)

$\ell(x)$ = #lvls occupied by x $\Rightarrow$ Height of the SKIP-LIST: $\max_x (\ell(x))$

$\mathbb{P}(\ell(x) \geq \ell)$ GIVEN = prob. of ℓ tosses $= \left(\frac{1}{2}\right)^{\ell}$ = x occupies at least ℓ levels

$$\mathbb{E}[\max_x \ell(x)] \overset{\text{def. of avg}}{=} \sum_{\ell \geq 1} \mathbb{P}(\ell \geq \ell) = \sum_{\ell=1}^{\lg n} \underbrace{\mathbb{P}(\ell \geq \ell)}_{\leq 1 \text{ def of prob :)}} + \sum_{\ell=\lg n+1}^{+\infty} \mathbb{P}(\ell \geq \ell) \leq \log n + \sum_{\lg n+1}^{+\infty} \mathbb{P}(\ell \geq \ell)$$

$\mathbb{P}(\ell(x) \geq \ell) = \frac{1}{2^{\ell}}$ is for one item only

$$\Rightarrow \mathbb{P}(\ell \geq \ell) = \mathbb{P}(\exists x : \ell(x) \geq \ell) \underset{\text{UNION BOUND}}{\leq} \underbrace{\mathbb{P}(\ell(x_1) \geq \ell)}_{(1/2)^{\ell}} \vee \underbrace{\mathbb{P}(\ell(x_2) \geq \ell)}_{(1/2)^{\ell}} \vee \dots \vee \mathbb{P}(\ell(x_n) \geq \ell) \underbrace{+ \dots + \left(\tfrac{1}{2}\right)^{\ell}}_{= \frac{n}{2^{\ell}}}$$

$$\Rightarrow \lg n + \sum_{\ell \geq \lg n+1} \frac{n}{2^{\ell}} \quad \left[\underset{\lg n+1}{\ell} = \lg n + \underset{1}{\ell'}\right] = \lg n + \sum_{\ell' \geq 1} \frac{n}{2^{\lg n + \ell'}} = \lg n + \sum_{\ell' \geq 1} \frac{\cancel{n}}{\cancel{n} * 2^{\ell'}}$$

$$= \log n + \sum_{\ell' \geq 1} \underbrace{\frac{1}{2^{\ell'}}}_{= \left(\frac{1}{2}\right)^{\ell'}} = \lg_2 n + 1$$

$$\Rightarrow \mathbb{P}(\ell(x) > c \cdot \log n) = \left(\frac{1}{2}\right)^{c \cdot \lg n} = \frac{1}{(2^{\lg n})^c} = \frac{1}{n^c}$$

BigData = Very high prob.

② Establish the search time

#steps → + #steps ← we just proved that on avg it's $\Theta(\lg n)$

look at it in reverse: I go LEFT iff there's no vertical pointer → $\boxed{\cancel{\uparrow} \; n}$ → = prob = $\frac{1}{2}$ $\Rightarrow$ on avg = 2

$\Rightarrow$ 2 log n on avg $\Rightarrow$ $\Theta(\log n) + \Theta(\log n) \to$ cost search $= \Theta(\log n)$

NOTE merge ∧ split are very easy, just put/remove a +∞ / −∞ in a spot

NOTE no locality of reference because it's a list

2nd Part

11/11

Locality Preserving Front Coding

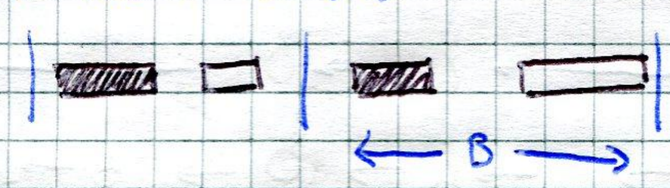
solves the problems of FRONT CODING

strings of variable length that share L

if I ask the content of a string

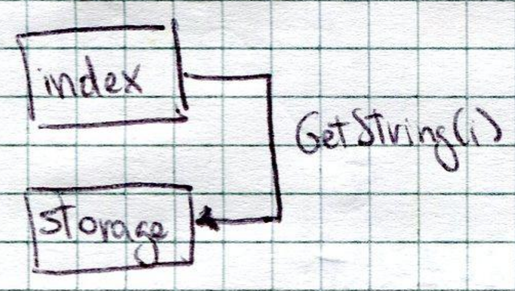
⇒ reconstruct it → worst case: ALL structure

divide in Blocks



tradeoff: small ⇒ fast but save more

Idea:



① constant Time access, no compression, int. direct access

② take $i \bmod B$ (?)

③ cost decompressing $\approx \frac{|s|}{B}$ (optimal)

⇒ exploit the structure!

Theo ∃ a storage method scheme that achieves space occupancy $(1+\epsilon)FC(D)$ "FrontCoding of set D"

∧ supports the string decompression of a string S in $O(\frac{|s|}{\epsilon B})$ I/Os

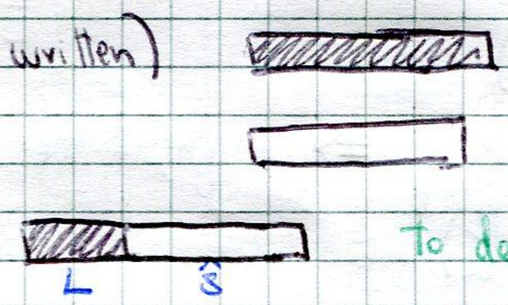
D = the entire set

DIM

Pick a string copied (= fully written)

and another string s

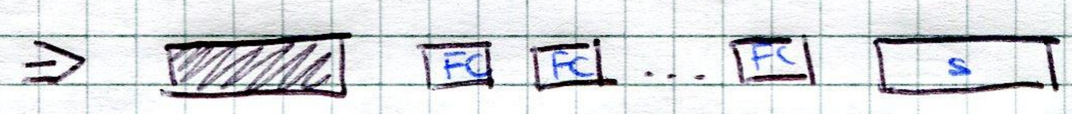
Now, compress s



To decompress go back to the start)

If the distance is $\leq c \cdot |s|$ ⇒ FC

o/w don't ≡ fully copy



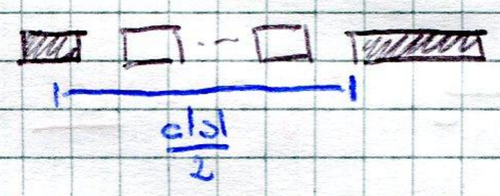
- $O(\frac{|s|}{\epsilon B})$ obvious

- space occupancy: 2 possibilities, the string is either FC or fully copied

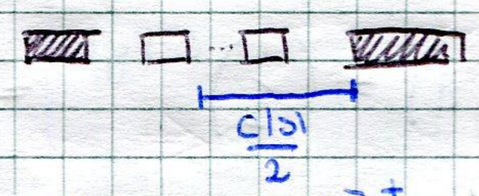
  ⇒ take the space of the copied string, because $\boxed{FC}\boxed{FC} \in (1+\epsilon)FC$

  ⇒ "$+\epsilon FC$" is the space of fully copied strings

⇒ CROWDED



UNCROWDED



scheme $[\text{uncrowded} \cdot (\text{crowded})^+]^+$

1 U, many C and repeat

space paid:

$$\sum_{i \geq 1} |w_i|$$

note: the $\boxed{FC}$ aren't important, they $\in (1)FC$

$$= \underbrace{|w_1|}_{\text{uncr.}} + \sum_{i>1} \underbrace{|w_i|}_{\text{cr.}}$$

$$|s'| \geq \frac{c \cdot |s|}{2} \Rightarrow |s| \leq \frac{2|s'|}{c}$$

In our context $c > 2 \Rightarrow s \leq s' \Rightarrow$ the length decreases

fully copied strings, we are sure is less than ∞ :)

so the $|w_1| + \sum_{i>1} |w_i| \leq |w_1|\infty + \sum_{i>1} \left(\frac{2}{c}\right)^{i-1} |w_1|$

$$= \sum_{i=1}^{\infty} \left(\frac{2}{c}\right)^{i-1} |w_1| = \underbrace{\sum_{i=0}^{\infty} \left(\frac{2}{c}\right)^{i}}_{\frac{c}{c-2}} |w_1|$$

$\Rightarrow$ the first one dominates the space

all the others are exponentially decreasing

Now take first uncrowded of each group and sum them

$$[\underset{U}{w_1}, \dots, \underset{C}{w_2}, \dots, \underset{C}{w_3}, \dots] \rightarrow \sum |w_i| \leq |w_1| \cdot \frac{c}{c-2}$$

$$\sum_{s=1}^{\cdot} \left(\underbrace{|w_1^s|}_{\text{the } s\text{-th sequence}} \cdot \frac{c}{c-2}\right) = \frac{c}{c-2} \cdot \sum_{s=1} |w_1^s|$$

length 1st string of sequence s. It's uncrowded!

$\Rightarrow$ I have a lot of FC characters (at least) $> \frac{c \cdot |s|}{2}$

FC - chars $\geq \frac{c \cdot |s|}{2}$

$|w_1^s| \leq |s| \leq \frac{2}{c}$ FC-chars preceding w_1^s

$$\frac{c}{c-2} \sum_{s=1} |w_1^s| \leq \frac{2}{c} \sum_{s} (\text{FC-chars preceding the string } w_1^s)$$

$$= \frac{2}{c-2} \sum_{s} (\text{FC-chars preced. } w_1^s) \leq \frac{2}{c-2} FC(D)$$

tot. # FC chars $\equiv FC(D)$

tot. occ. space $FC(D) + \underbrace{\frac{2}{c-2} FC(D)}_{\text{copied}}$

ε $c > 2 \Rightarrow$ constant

ex aabb aaabbb abbbc abbbd abbbe abc

① FC <0,aabb> ; <2,abbb> ; <1,bbbc> ; <4,d> ; <4,e> ; <2,c>

② Assume $c=1$, count prev. chars and digit to check the backward copy

<0,aabb> ; length = 6 ≡ come back 6-times <2,abbb> ; no FC
5 4 3 2 1 5<6 ⇒ FC it

<0,abbbc> ; <0,abbbd> ; <0,abbbc> ; <0,abc>

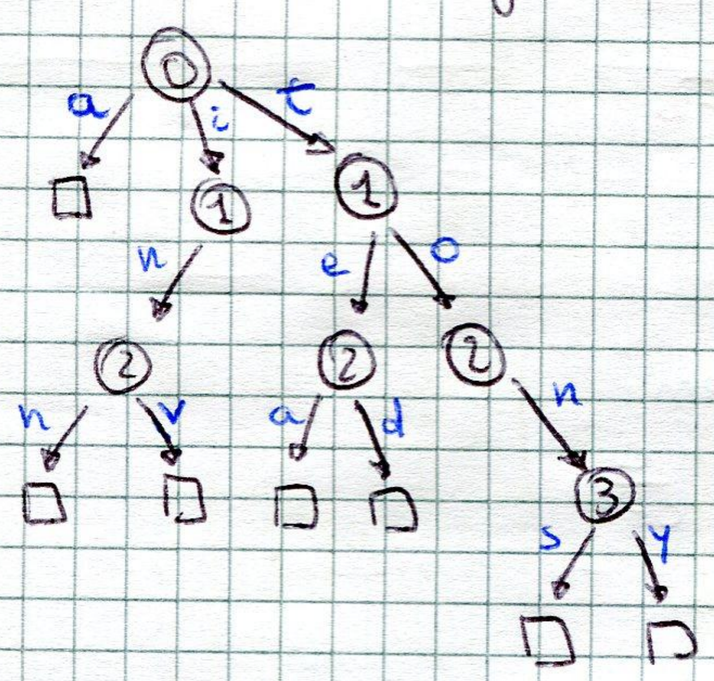
* PAGU LESSON *

10/11

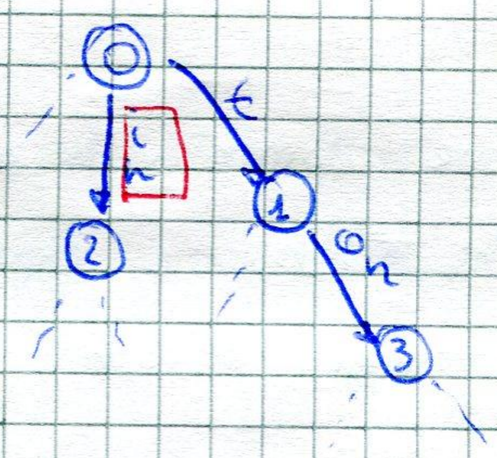
Compacted Trie

A Tree where every path represents a string (edges sorted alphabetically)

UNCOMPACTED (edge = 1 char)



COMPACTED (drop unary paths)



DON'T represent explicitely

substitute [] with a pair

position 10 11 12 13 14 15

a #inn #inv #...

don't represent "inn", use <12,2> as a pointer to in

⇒ independent on the length of the string ≡ constant space

If n = #leaves (= #strings)

The #internal nodes is O(n) #edges ≤ #leaves + #nodes − 1 < 2n − 1

⇒ space occupied proportional to n space = O(n) [LARGE CONSTANTS 2(1/ε) J]

Dividing storage from indexing

strings are sometimes we work on this

To store a string, LocPresFC ≡ gzip/decompress s in opt

#steps to search = |string|

Prefix/Lexicogr. search cost = O(|P| + occ)

(P) path

can't keep in internal memory

⇒ Patricia Trie

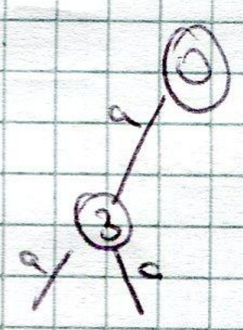
Idea DROP substrings

Store → HashTable

so aba ⇒ a , bcbc ⇒ b ...

⇒ losing infos inside the Trie

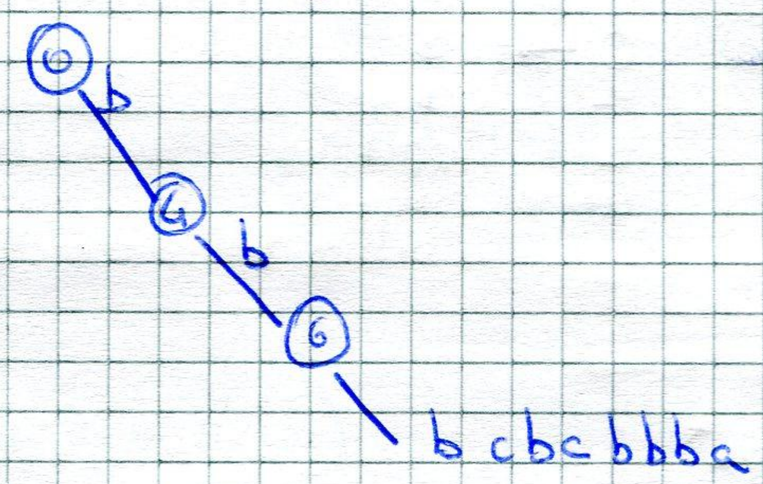
⇒ Search ≡ OPTIMISTIC



⇒ **Phase 1** go down (get a leaf)

0 I/Os because all in internal memory! Given leaf, access disk ∧ compare

P = bcbabc ⇒



totally wrong! ⇒ ②

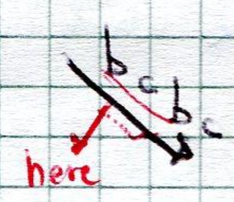
Phase 2 Found a mismatch ⇒ Longest Common Prefix (LCP) ∀ step → 1 mismatch max
n steps ⇒ O(n)

at most $\frac{|P|}{B}$ I/Os

we found $\underbrace{bcba}_{LCP}$ c ← mismatch char

we see that a < c ≡ pattern P is < than the string

Phase 3 go up the tree for "3"



Substring Search?

T = MISSISSIPPI$ P = SSI ⇒ return 3, 6
1 2 3 4 5 6 7 8 9 10 11

Linear scan = O(|P| · T)

Idea Take all the suffixes of T

POS	SUF
12	$
11	i$
10	pi$
9	ppi$
8	ippi$
7	sippi$
6	ssippi$
5	issippi$
4	sissippi$
3	ssissippi$
2	ississippi$
1	mississippi$

NOTE P occurs in T at position ② iff P prefixes T[i,n]

"ssi" is the prefix of ssippi ⑥
∧ ssissippi ③

⇒ substring search into a prefix search!

but space is quadratic (bleah)

⇒ SORT SUFFIXES

SA = Suffix Array

SA	SUF
12	$
11	i$
8	ippi$
5	issippi$
2	ississippi$
1	mississippi$
10	pi$
9	ppi$
7	sippi$
4	sissippi$
6	ssippi$
3	ssissippi$

O(n) space (1 integer per position)

O(n log n) bits

```
SABuild(char* T, int n, int *SA)
    for(i=0; i<n; i++)
        SA[i] = T+i
    qsort(SA, n, sizeof(char*), suffix_cmp)
                                 ||
                                 strcmp(*p, *q)
```

Search with binsearch then jump to memory.

1 → MISSISSIPPI, M < S → search [10 3] —

... → 7 = sippi → ssissip $\xrightarrow{\text{below}}$ 6 ssi = ssi

O(p · log n + occ · p)

[NOTES: p$ smallest p# largest]

Tipically we LCP between suffixes

⇒ simply the

0 →	$
1	i$
1	ippi$
4	issippi$
...	ississippi$

21/11

ANANAS$ example

SUFFIX-TREE

compacted trie of all suffixes of T

Linear time to build (#edges < 2n)

with SA & LCP array you can build T and viceversa

SA = visit leaves

LCP = in order visit T

Text Mining ① Check ∃ repeated substring of length ≥ L

scan LCP : if ∀i ∈ LCP i < L ⇒ no, yes o/w

② Check ∃ substring of length L which occurs at least C times

- in ST find a node x ≥ L : #leaves ≥ C
- in LCP find a contiguos sequence of numbers ≥ L ∧ they're ≥ (c−1)

25/11

KRUSKAL (UNDIRECTED)

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

① Assign ints to nodes via DFS → adj. list sorted by destination ID

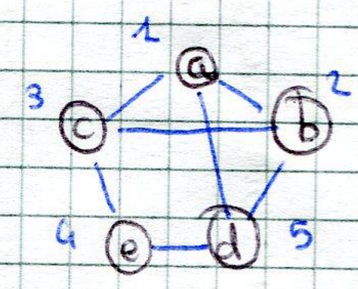
② Assign weights: given (x,y)

→ cost(x,y) = 2 * rank(x) * rank(y) mod 7

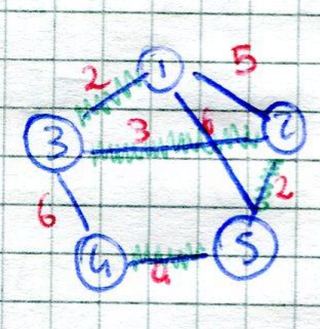
rank(x) = pos. of x in alph. order, from 2

	a	b	c	d	e
RANK	2	3	4	5	6

a →	b c d
b →	~~a~~ c d
c →	a ~~b~~ e
d →	a b e
e →	c d



⇒ rank



② ③ ① ordering

<x, y, c>

<1, 3, 2> ✓

<2, 5, 2> ✓

<2, 3, 3> ✓

<4, 5, 4> ✓

<1, 2, 5>

<1, 5, 6>

<3, 4, 6>

STOP when n. (leave to loop)

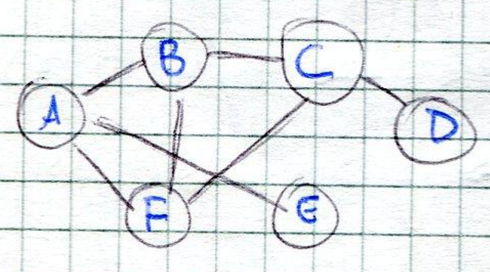
note KRUSKAL is semi-external (≡ edges → disk, nodes → internal memory)

⇒ very good when #nodes < M

cost sort(|E|) + ~~scan(|E|)~~ $\frac{|E|}{B}$ (if > M → |E|)

PRIM (UNDIRECTED)

(a,b,2) (a,f,4) (a,e,6) (b,c,1) (b,f,3) (c,d,3) (c,f,5)



build MST by expanding component

kruskal "merges" them

cost $\sum_{n} 1 + \frac{adj[u]}{B} = n + \frac{|E|}{B}$

|V|

ops	A	B	C	D	E	F
START	∅	∞	∞	∞	∞	∞
Extract A	•	2_A			6_1	4_A
Ex B		•	1_B			3_B
Ex C			•	3_C		
Ex D				•		
Ex F						•
Ex E					•	

SYBEIN'S ALGORITHM

π: random permutation of nodes

Q: priority queue

for $e=(u,v) \in E \rightarrow$ Q.insert (minimum $\{\pi(u), \pi(v)\}$, maximum $\{\pi(u), \pi(v)\}$, $c(e)$, u, v) ① ③ ②

current $= \emptyset$

while $Q \neq \emptyset$

extract the minimum (u, v, c, u_0, v_0)

if current $\neq u$

if $u == n-M+1$ BREAK //fits in internal memory

Q.deleteMin (redundant!)

print (u_0, v_0)

(current, relinkTo) $= (u, v)$

elif $v \neq$ relinkTo

Q.insert (min $\{v$, relinkTo$\}$, max $\{v$, relinkTo$\}$, c, u_0, v_0)

Apply semi-external Krusnal algorithm

exercises

27/11

INTEGER CODING

$S = S_1, S_2, \dots, S_n$ we assume it's nondecreasing $S_i \in \mathbb{N}$

↳ Gap-coding: $S' = S_1, (S_2 - S_1), (S_3 - S_2), \dots, (S_n - S_{n-1})$

if there are negative numbers $\rightarrow x \geq 0 \rightarrow 2x$ $x < 0 \rightarrow -2x+1$

for the compressed number length → FIXED length → memory word 4/8 bytes
→ $1 + \lfloor \log_2 \max\{S_i\} \rfloor$ bits

↓

VARIABLE length $\forall i$ $1 + \lfloor \log_2 S_i \rfloor$ bits

SHANNON (source of items)

Σ = alphabet of items

$$H = \sum_{\sigma \in \Sigma} \mathbb{P}(\sigma) \cdot \log_2 \frac{1}{\mathbb{P}(\sigma)}$$

entropy of the source

Theo Given c, the optimal $l(c) = \log_2 \frac{1}{\mathbb{P}(c)}$

ex $\Sigma = \{a, b, c\}$ $\frac{1}{2}$ $\frac{1}{4}$ $\frac{1}{4}$ 0 10 11

$H = \frac{1}{2} * 1 + \frac{1}{4} * 2 + \frac{1}{4} * 2 = 1.5$ bits

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

Min when $H = \emptyset$? either $\mathbb{P}(\sigma) = 0$
$\log_2 \frac{1}{\mathbb{P}(\sigma)} = \emptyset \rightarrow \mathbb{P}(\sigma) = 1$ = stupid source, no infos!

when $H = \log_2 |\Sigma|$? (max)

$\mathbb{P}(\sigma) = \frac{1}{|\Sigma|}$ ≡ uniform distribution

$\Leftarrow H = \sum_{\sigma \in \Sigma} \frac{1}{|\Sigma|} \cdot \log_2 |\Sigma| = \log_2 |\Sigma|$

Average Code length

$$\mathbb{E}[\text{code}] = \sum_{\sigma \in \Sigma} \mathbb{P}(\sigma) \cdot \underbrace{L(\sigma)}_{\text{len in bits of the code for } \sigma} \quad \text{in bits}$$

$$H = 1.5 \quad \begin{cases} 1/2 & a \to 10 \\ 1/4 & b \to 01 \\ 1/4 & c \to 000 \end{cases} \qquad \mathbb{E}[\text{code}] = 1/2 \cdot 2 + 1/4 \cdot 2 + 1/4 \cdot 3 = 2.25 \text{ bits}$$

note $L(\sigma) \approx \log_2 \frac{1}{\mathbb{P}(\sigma)}$ — the more frequent, less bits used (e.g. $\mathbb{P}(\cdot) = 1 \to$ no bits)

GOLDEN RULE OF DATA COMPRESSION

UNARY CODE

$U(x)$ — $U(1) = 1$, $U(2) = 01$, $U(3) = 001$

$$U(x) = \underbrace{0 \ldots 0}_{x-1} 1$$

$$|U(x)| = x = \log_2 \frac{1}{\mathbb{P}(x)}$$

$$x = \log_2 \frac{1}{\mathbb{P}(x)}$$

$$2^x = \frac{1}{\mathbb{P}(x)} \quad \mathbb{P}(x) = 2^{-x}$$

GAMMA CODE

$$\gamma(x) = \underbrace{0 \ldots 0}_{L-1} \underbrace{bin(x)}_{L} = 2L - 1 \text{ bits}$$

$2 \log_2 x - 1$ bits to encode

SLOW IN DECODIN

$$2 \log x - 1 = \log_2 \frac{1}{\mathbb{P}(x)} \qquad \cancel{\log} \frac{x^2}{2} = \cancel{\log_2} \frac{1}{\mathbb{P}(x)} \qquad \frac{x^2}{2} = \frac{1}{\mathbb{P}(x)} \Rightarrow \mathbb{P}(x) = \frac{2}{x^2}$$

DELTA CODING

$$\delta(x) = \underbrace{\gamma(L)}_{2 \log_2 L - 1} \underbrace{bin(x)}_{L} \qquad L = \log_2 x$$

$$\delta(7) = \gamma(3)\, 111 = 011111$$

$$\delta(10) = \gamma(4)\, 1010 = 001001010$$

optimal $\mathbb{P}(x) \approx \frac{1}{2 x \log^2 x}$

RICE CODING (parametrized)

$$R_k(x) = \underbrace{q+1}_{\text{unary part}} \; \underbrace{r}_{\text{binary part}} \qquad \equiv \text{rest} \wedge \text{quotient}$$

$$q = \left\lfloor \frac{x-1}{2^k} \right\rfloor \qquad r = x - 1 \mod 2^k$$

$q+1$ because it can be 0

$$R_4(83) = \left.\begin{array}{l} q = \left\lfloor \frac{82}{16} \right\rfloor = 5 \xrightarrow{+1} 6 \Rightarrow 000001 \\ r = 2 \qquad bin_4(2) \Rightarrow 0010 \end{array}\right\} 00000\,10010$$

note $2^k \approx \frac{\ln 2}{P}$ — optimal, P prob. of geometric d

PFORDELTA (start, b)

3 ~~7~~ 2 4 5 2 3 ...

start=2 → 1 5 0 2 3 0 1 ...

b tells us the fixed length $\{0, \ldots, 2^b - 2\}$, $2^b - 1$ escape

ex $b = 2 \Rightarrow$ configs $\{0, 1, 2\} \wedge 3$ ESCAPE

1	5	0	2	3	0	1	...
01	11	00	10	11	00	01	

OBS $b \nearrow$ = extra ↓ = waste

VARIABLE BYTE (SC dense code)

Given b (len in bits), $s+c=2^b$

The first s numbers $\{0 \dots s-1\}$

$s \dots s \cdot c$

$s \cdot c \dots s \cdot c^2$

...

continuers, stoppers

$\boxed{s}$

$\boxed{c}\ \boxed{s}$

$\boxed{c}\ \boxed{c}\ \boxed{s}$

$b=3 \quad 2^b=8 \quad s=4, \ c=4$

$S=\{000, 001, 010, 011\}$

$C=\{100, 101, 110, 111\}$

0	000
1	001
2	010
3	011
4	100 000
5	" 001
6	" 010
7	" 011
8	101 000
...	

...

Interpolative Code (non parametric, slow, very compressed)

$s_1, \dots, s_l, \dots, s_r, \dots, s_n \quad [s_i < s_{i+1}]$

Quintuple $\langle l, r, n, low, hig \rangle$

INIT $l=1 \quad r=K \quad n=K \ (=r-l+1) \quad low=s_1 \quad hi=s_K$

take middle elem

$m = \lfloor \frac{r+l}{2} \rfloor \rightarrow s_m$

~~Then BinaryCode(Sm, low, hig)~~

Then BinaryCode $(s_m, low+m-l, hi+m-r)$

recursion on $s_l \dots s_{m-1}$ using $\langle l, m-1, m-l, low, s_{m-1} \rangle$

$s_{m+1} \dots s_r$ using $\langle m+1, r, r-m, s_{m+1}, hi \rangle$

BinaryCode$(x, a, b) \quad x \in \{a, \dots, b\}$

$bin(x-a)$ in $\lceil \log_2 b-a+1 \rceil$ bits

es. BinCode(7, 3, 10)

$bin(4)$ in $\lceil \log_2 \underset{3}{10-3+1} \rceil \Rightarrow 100$

es $s_1 \quad s_6 \quad s_{12}$

1 2 3 5 7 9 11 15 18 19 20 21

$l=1 \quad r=12$

$n=12 \quad low=1 \quad hi=21$

BinaryCode$(9, 6, 15) \rightarrow 3$ in 0011

$\langle 1, 5, 5, 1, 8 \rangle$ → recurse on 3

$\langle 7, 12, 6, 10, 21 \rangle$ → recurse on 18

ELIAS-FANO

$S_1 \dots S_n < u$

$w = \lceil \log_2 u \rceil \quad u$ is the 2^x closer to S_n

$\left[l = \lfloor \log_2 \frac{u}{n} \rfloor = w-h \right] \quad h = \lfloor \log_2 n \rfloor$

1	00	01
3	00	11
5	01	01
11	10	11
12	11	00
15	11	11
	h	l

$L = 01 \ 11 \ 01 \ 11 \ 00 \ 11 \leftarrow l \cdot n$

$H = 110 \ 0 \ 1 \ 0110 \leftarrow \#0: 2^h = 2^{\log_2 n} = n \Rightarrow 2n$

$\#1: n$

∀ change in 1st part put a 0

∀ change put a 1

⇒ Implement Access Retrieve the i-th value

① get i-th block in L

② $Select_1(i)$ returns the position of the i-th 1, starting from 1

$Select_1(4)$? ⇒ 6

Then $Select_1(i) - i$

TOTAL $\underset{|H|}{2n} + \underset{|L|}{n \log_2 \frac{u}{n}}$

$= n(2 + \log_2 \frac{u}{n})$

⇒ EF. uses 2 bit more than optim

DIN

0101010000001

$\mathbb{P}(1) = \frac{n}{u} \quad \mathbb{P}(0) = 1 - \frac{n}{u}$

$H = p(1) \cdot \log_2 \frac{1}{p(1)} + p(0) \cdot \log_2 \frac{1}{p(0)}$

$\geq p(1) \cdot \log \frac{1}{p(1)}$

$= \frac{n}{u} \cdot \log_2 \frac{u}{n} \rightarrow n \log_2 \frac{u}{n}$

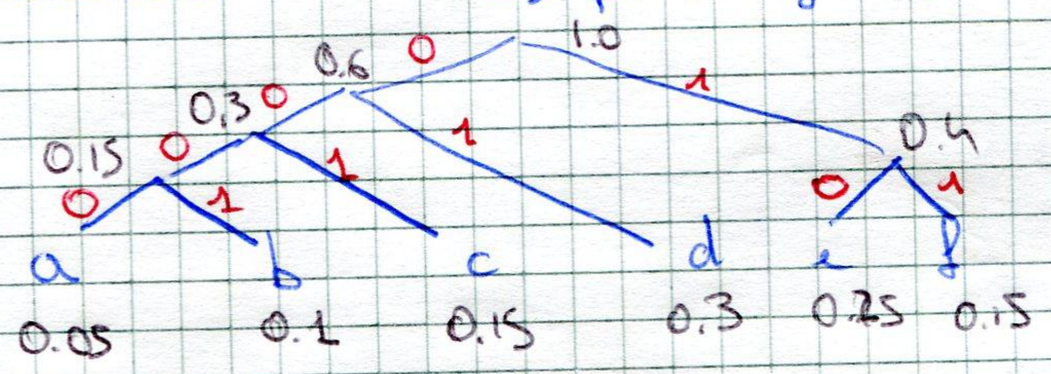
3/12

STATISTICAL COMPRESSORS (greedy strategy)

HUFFMAN CODE

Semi static model for the frequences

$T = abaabaac \Rightarrow f(a) = 5/8 \quad f(b) = 2/8 \quad f(c) = 1/8$



Huffman Tree

Pick the pair of the least weighted nodes

Then CODEWORDS path root → leaf with 0s to the left, 1s to the right

$cw(a) = 0000 \quad \ldots \quad cw(f) = 11$

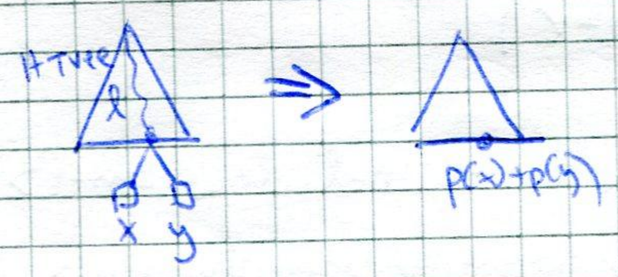
$$L_{Huff} = \sum_{\sigma} p(\sigma) * |cw(\sigma)| \geq H$$

(avg length distinct)

it's also the avg depth of the Huffman Tree

Theo Huffmann is optimal for any prefix-free code

Lemma 1



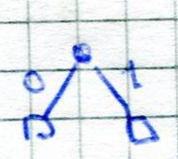
$$L_T^{(avg)} = \text{avg depth of Tree} = \sum_{\sigma} p(\sigma) * d(\sigma)$$

$$= \sum_{\sigma \neq x,y} p(\sigma) d(\sigma) + p(x) \underset{l+1}{d(x)} + p(y) \underset{l+1}{d(y)}$$

$$= \sum_{\sigma \neq x,y} p(\sigma) d(\sigma) + l(p(x) + p(y)) + (p(x) + p(y))$$

$$= L_{RT} + p(x) + p(y)$$

Lemma 2



$\Sigma = \{a, b\}$ only 2 nodes $\Rightarrow$ Huffmann is optimal

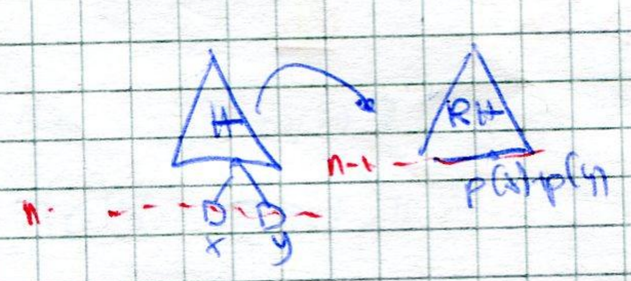
Lemma 3

Given a probabilistical distribution, if is an optimal code $\wedge$ x,y are the right

$\Rightarrow$ they're the deepest $\wedge$ merged together

D/H by induction on the #leaves

Assume Huffmann is optimal for n-1 symbols $\Rightarrow$ Prove it for n



$$L_H \overset{①}{=} L_{RH} + p(x) + p(y) \leq L_{T_{n-1}} + p(x) + p(y)$$

it's on n-1 symb! $\Rightarrow$ it's optimal

T qualsiasi albero con n-1 simbol

optimal code on n symbols

If n symbols, T_{opt}. Because of ③ I merge deepest leaves $\Rightarrow L_{T_{opt}} = L_{RT_{opt}} + p(x) + p(y)$

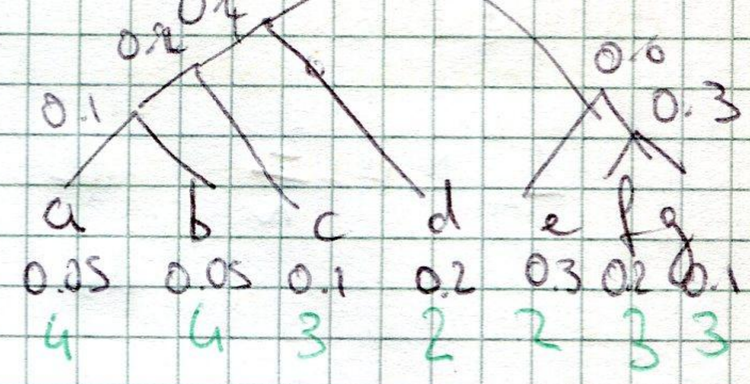
$$\geq L_{RH} + p(x) + p(y) = L_H \geq L_{T_{opt}}$$

$\Rightarrow L_H$ optimal

CANONICAL HUFFMANN

① Do Huffman Code ∧ count the depth leaf

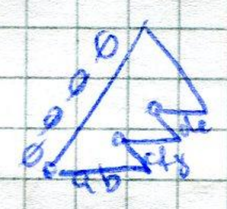
② Create table (symbol matrix)



[1, max depth]

depth	symb	num
1	/	∅
2	d, e	2
3	c, f, g	3
4	a, b	2

③ Transform ↗ in



in l bits!

l	fc	$bin_l(fc)$
4	∅	0000
3	$\frac{0+2}{2}=1$	001
2	$\frac{1+3}{2}=2$	10
1	$\frac{2+2}{2}=2$	impossible!

↖ happen always!

Alg: $fc[\overset{\text{depth max}}{max}]=\emptyset$

for all ($l=max-1$; $l\geq 1$; $l--$)

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

Decoding

```
v=next_bit()
while (v< fc[l])
    v= 2v+next_bit()
    l++
return symb [l, v - fc[l]]
```

starts from 0

01011 ...

$l=1$, $v=0$ $\quad v<\ldots$ ✓

$l=2$, $v=1$ $\quad 1<2$ ✓

$l=3$, $v=2$ $\quad 2<1$ ×

⇒ symb[3, 2 - 1] ⇒ return symb[3, 1]

I° simbolo livello 3 (from 0) ⇒ f

note no pointer jumping

fc ∧ symb typically fits in cache

$H \leq L_{Huff} < H+1$

ARITHMETIC CODO

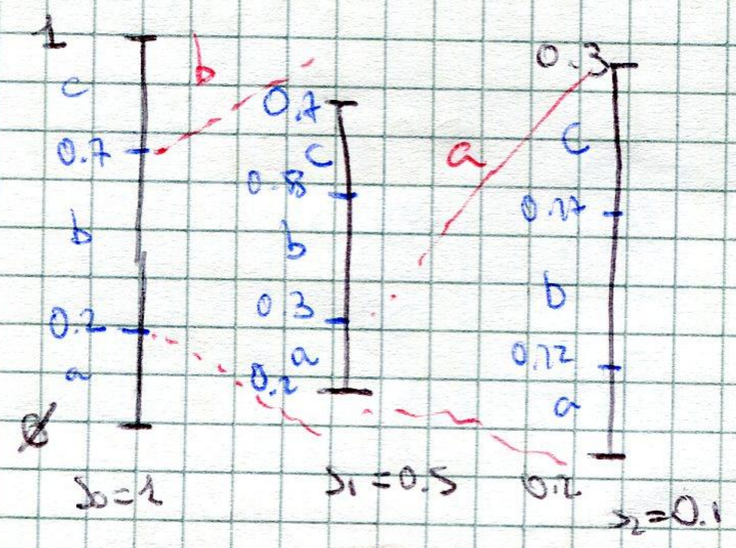
$L_{AM} \simeq H + \frac{2}{n}$ iff with INFINITE PRECISION ARITHMETIC

else $\simeq H + \frac{1}{100}$

$\Sigma = \{a, b, c\}$ $T = bac$

$\mathbb{P}$ 0.2 0.5 0.3

$f(a) = \emptyset$ $f(b) = 0.2$ $f(c) = 0.7$ $f(c_i) = \sum_{j=1}^{i-1} \mathbb{P}(c_j)$ sum of all probs before that



for the last char you can just pick the middle elem

$0.27 - 0.3 \Rightarrow 0.285$

$[0.27, 0.3)$

range $\equiv s_1 = 0.7 - 0.2 = 0.5$

$\Rightarrow \mathbb{P}(a) \cdot s_1 \Rightarrow 0.1$

$\mathbb{P}(b) \cdot s_1 \Rightarrow 0.25$

$\mathbb{P}(c) \cdot s_1 \Rightarrow 0.15$

2nd step $0.2 \cdot 0.1 = 0.02$

0.05 0.03

$\Rightarrow T = 0.28$

note $s_n = \prod_{i=1}^{n} \mathbb{P}(T[i])$

dyadic fraction $\frac{v}{2^k}$ it's easily plottable

$\Rightarrow$ v in binary in k bits

$\frac{3}{16} \Rightarrow 3$ in 4 bit $\Rightarrow .0011$

2^4

FractionalEncode(x)

```
x = 2x
if (x < 1) print ∅
else x = x - 1 print 1
while x ≠ 0
```

you can stop emit digits when I've emitted d digits

$\equiv \lceil \log_2 \frac{2}{s_n} \rceil$

$x = 0.b_1 b_2 \ldots b_d b_{d+1} \ldots$ (bits) 0.28

$\hat{x} = 0.b_1 b_2 \ldots b_d$ Trunc after d

Have to guarantee that $x - \hat{x} \leq \frac{s_n}{2}$

$l_n + s_n$

$l_n + \frac{s_n}{2}$ can be represented

l_n

$x - \tilde{x}$ is represented as $0 \ldots 0\, b_{d+1} \ldots$

$$\Rightarrow \sum_{i=1}^{\infty} b_{d+i} * 2^{-(d+i)} = \sum_{i=1}^{\infty} 2^{-(d+i)} = 2^{-d} \cdot \underbrace{\sum_{i=1}^{\infty} 2^{-i}}_{=1} = 2^{-d}$$

BITS

b_{d+i}-esimo is a 1

difference between entire representation and truncated one

why $d = \lceil \log_2 \frac{2}{S_n} \rceil$

$$2^{-d} = 2^{-\lceil \log_2 \frac{2}{S_n} \rceil} = \frac{1}{2^{\lceil \log_2 \frac{2}{S_n} \rceil}} \leq \frac{1}{2^{\log_2 \frac{2}{S_n}}} = \frac{1}{\frac{2}{S_n}} = \frac{S_n}{2}$$

c.v.d.

$d \overset{?}{\approx} H$?

$$\lceil \log_2 \frac{2}{S_n} \rceil \leq 1 + \log_2 \frac{2}{S_n} = 1 + \log_2 2 - \log_2 S_n = 2 - \log_2 S_n$$

$$= 2 - \log_2 \prod_{i=1}^{n} P(T[i])$$

log of π is sum of the log

$$= 2 - \sum_{i=1}^{n} \log_2 P(T[i])$$

$$= 2 - \sum_{\sigma} occ(\sigma) * \log_2 P(\sigma) =$$

$-\log(a) = \log(a)^{-1} = \log \frac{1}{a}$

$$= 2 + \sum_{\sigma} occ(\sigma) * \log_2 \frac{1}{P(\sigma)}$$

$$= 2 + n \sum_{\sigma} \underbrace{\frac{occ(\sigma)}{n}}_{P(\sigma)} * \log_2 \frac{1}{P(\sigma)} = 2 + nH$$

$$\Rightarrow d \leq 2 + nH$$

DICTIONARY BASED COMPRESSORS

LZ77

window

aa caa cab | caaaab

phrase = #steps to ←, len, next char

<6, 3, a>

LZ-SS

<0, c> only next char

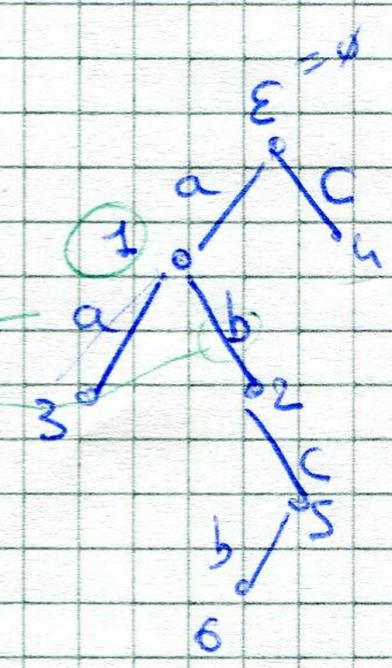
<d, l> only

4

0

LZ-78

T = a | ab | aa | c | ab | cab | cb

<0,a> <1,a> <0,c> <2,a> <5,b>

<1,b>