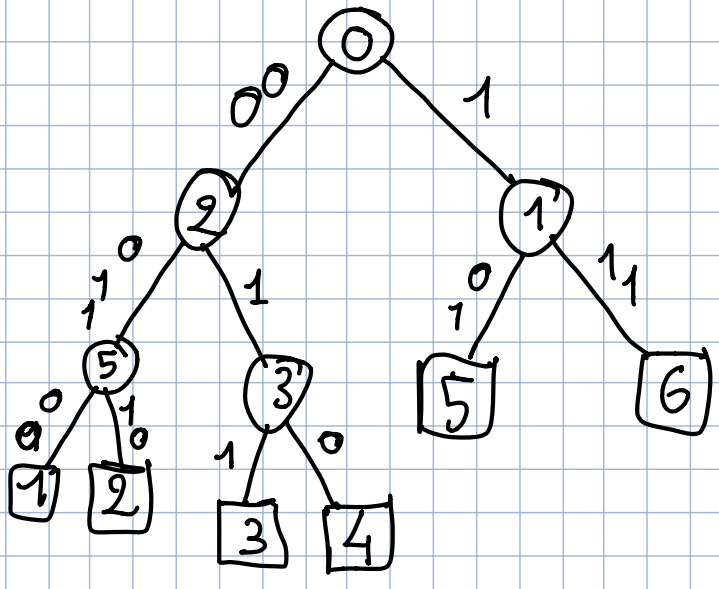
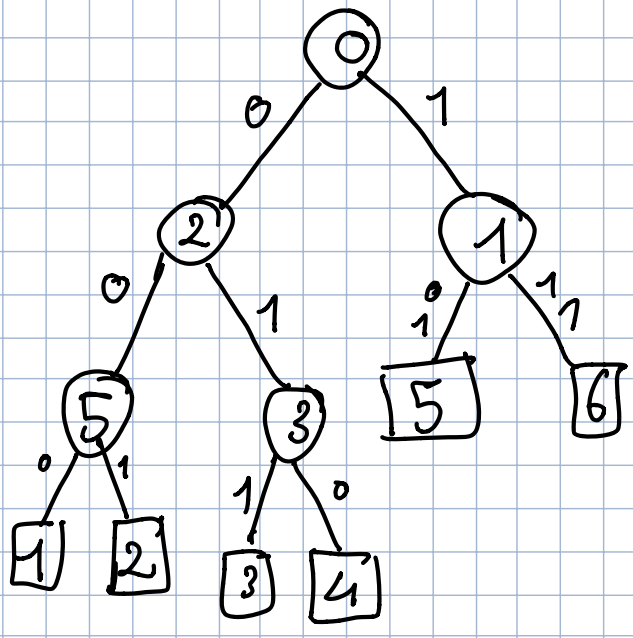
ESERCIZIO 1

$S = \{00001100, 00011110, 0011, 0010, 101, 111\}$

Compacted trie



Patricia trie



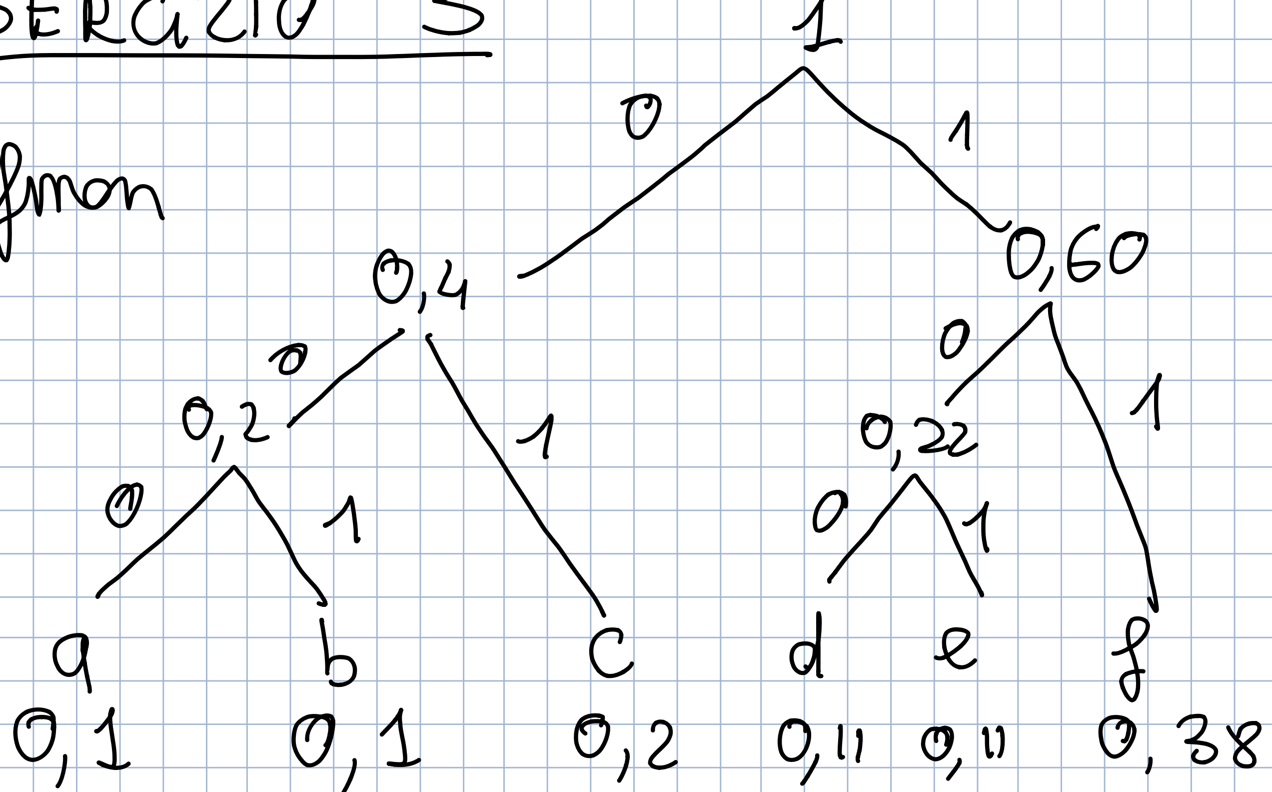
	± / 0	
Prefix Search con Compacted trie	$O\left(P + \frac{occ}{B}\right)$	occ = numero occorrenze stringhe prefisso di P.
Prefix Search con Patricia trie	$O\left(\frac{P}{B}\right)$	dove P = lunghezza prefisso da confrontare perché faccio il confronto con la stringa con cui ho il mismatch

Se vogliamo le stringhe va aggiunto $O\left(\frac{NoCC}{B}\right)$ dove

Nocc = lunghezza stringhe
da prendere

ESERCIZIO 3

Huffman



Canonical Huffman

num

1	2	3
0	2	4

Symb

1	2	3
/	c	a
	f	b
		d
		e

fC =

1	2	3
2	2	0
/	10	000

Decode: 10 000 1 0 11 0 11100

first 3 symbols:

① $V = 1 < fc[1]$ Si

$V = 2 < fc[1]$ No

$10 = C$

$Symb[2, 2-2] = [2, 0]$

② $V = 0 < fc[1]$ Si

$V = 0 < fc[2]$ Si

$V = 1 < fc[3]$ NO

$001 = symb[3, 1-0] =$

$001 = b$

③ $V = 0 < fc[1]$ Si

$V = 1 < fc[2] =$ Si

$V \geq 3 < \text{fc}[3]$ NO

$\text{Symb}[3, 3-0] = [3,3)$

$071 = l$

ESERCIZIO 4

Perfect Hash table è una hash table in cui la funzione hash è tale che $\forall x, y$ $h(x) \neq h(y)$.

Questa perfect Hash table è formata da una prima tabella e da una funzione h_1 che mappa i valori all'interno di un bucket. Poi abbiamo per ogni bucket una seconda hash table. Quindi in ulteriori hash table che hanno ognuna una funzione che non causa collisioni.

Lo spazio complessivo è $O(n)$.

Dim:

Dobbiamo dimostrare che $\sum_{j=1}^{m} n_j^2 < 2n$ dipende che $m = n$

Sappiamo che $a^2 = a + 2\binom{a}{2}$

Quindi

$$\sum_{j=1}^{m} n_j^2 = \sum_{j=1}^{m} n_j + 2\binom{n_j}{2} = \sum_{j=1}^{m} n_j + 2\sum_{j=1}^{m}\binom{n_j}{2}$$

$$n + 2\underbrace{\sum_{j=1}^{m}\binom{n_j}{2}}_{\hookrightarrow \binom{n}{2}\cdot\frac{1}{m}} = n + 2\binom{n}{2}\cdot\frac{1}{m} = n + 2\frac{n(n-1)}{2}\cdot\frac{1}{m}$$

$$= \boxed{n + n - 1 < 2n}$$

↳ perché è una universal hash function quindi il numero di collisioni è limitato da $\frac{|H|}{m}$.

ESERCIZIO 2

Multi Key Quick Sort:

Viene presa una stringa come pivot, e un intero i quindi confronto il pivot[i] con

le altre stringhe (il carattere i).
E divido in $S_<$, $S_=$, $S_>$.
Per ogni divisione una chiamata ricorsiva, ma i aumenta solo in $S_=$:

```
MultikeyQS(S, i) {
    p = stringa random di S
    S< = Stringhe minori di S[i]
    S> = Stringhe maggiori di S[i]
    S= = Stringhe uguali a ~
    MKQS(S<, i)
    MKQS(S=, i+1)
    MKQS(S>, i)
}
```

Costo in tempo: devo confrontare V stringa ds caratteri nel caso in cui capito in $S_=$, invece negli altri due casi fa delle chiamate su set che se il pivot

è detto bene allora mi fa una divisione in due parti uguali:

$O(d_{st} \lg n) \rightarrow$ n stringhe

$O(n(ds + \lg n)) = \underbrace{O(d + n \lg n)}$

come il lower bound

$S = [zoo, zorro, boa, bottle, abi, box, zio, hut]$

Scelgo il pivot: hut, $i = 1$

zoo
zorro
boa
bottle
abi
box
zio
hut

$S_< \{boa, bottle, abi, box\}$

$S_= \{hut\}$

$S_> \{zoo, zorro, zio\}$

E poi richiamo ricorsivamente su $S_<$, $S_>$ e $S_=$

nel caso di S_2 viene restituito direttamente "hit".