## Problem 1

i. Linear probing with $h(k) = (k+1) \bmod 5$

| | |
|---|---|
| 0 | 1 |
| 1 | |
| 2 | 21 |
| 3 | 16 |
| 4 | 3 |

ii. Cuckoo hashing with $h_1(k) = k \bmod 5$ and $h_2(k) = \lfloor k/5 \rfloor$

| | |
|---|---|
| 0 | 3 |
| 1 | 1 |
| 2 | |
| 3 | 16 |
| 4 | 21 |

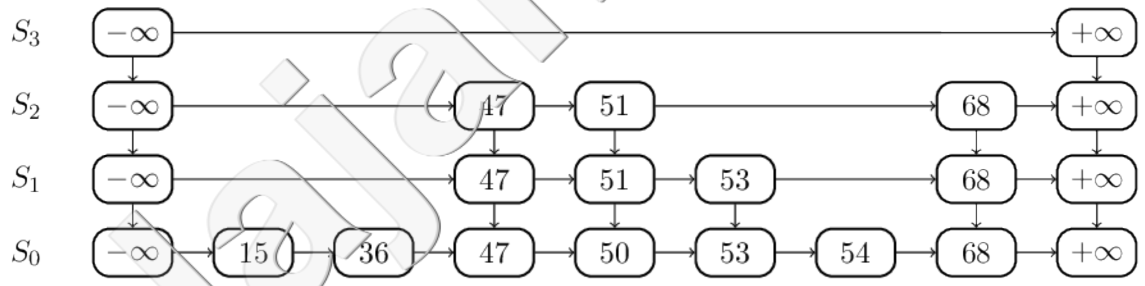
## Problem 2

a) Here, we give the list of all positions $p$ in $L$ in the algorithm of Slide 9 of Module 6.

Search($S$, 360): node $-\infty$ at level 4, node $-\infty$ at level 3, node 287 at level 3, node 287 at level 2, node 287 at level 1, node 287 at level 0. We accept if they add "node 360 at level 0".

Search($S$, 17): node $-\infty$ at level 4, node $-\infty$ at level 3, node $-\infty$ at level 2, node $-\infty$ at level 1, node $-\infty$ at level 0. We accept if they add "node 17 at level 0".

b)


$S_3$: $-\infty$, $+\infty$
$S_2$: $-\infty$, 47, 51, 68, $+\infty$
$S_1$: $-\infty$, 47, 51, 53, 68, $+\infty$
$S_0$: $-\infty$, 15, 36, 47, 50, 53, 54, 68, $+\infty$

c) In addition to $\{-\infty, +\infty\}$, $S_2$ should contain the keys with index

$$\{\lfloor n^{2/3} \rfloor, \lfloor 2\, n^{2/3} \rfloor, \ldots, \lfloor \lfloor n^{1/3} \rfloor n^{2/3} \rfloor\}.$$

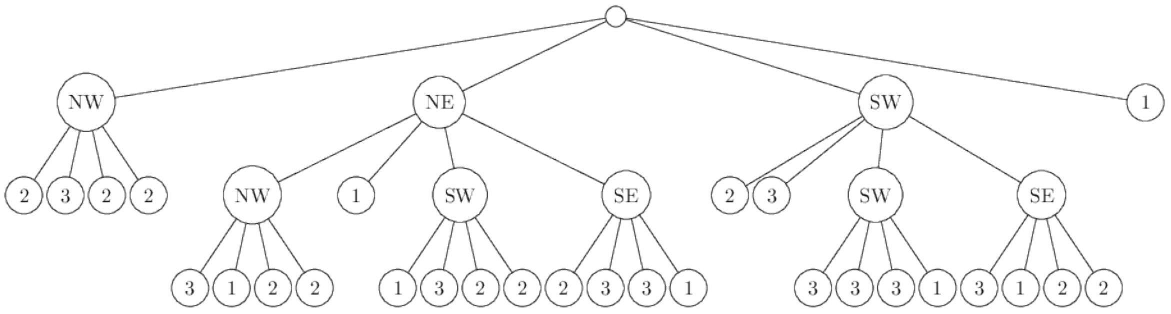
In addition to $\{-\infty, +\infty\}$, $S_1$ should contain the keys with index

$$\{\lfloor n^{1/3} \rfloor, \lfloor 2\, n^{1/3} \rfloor, \ldots, \lfloor \lfloor n^{2/3} \rfloor n^{1/3} \rfloor\}.$$

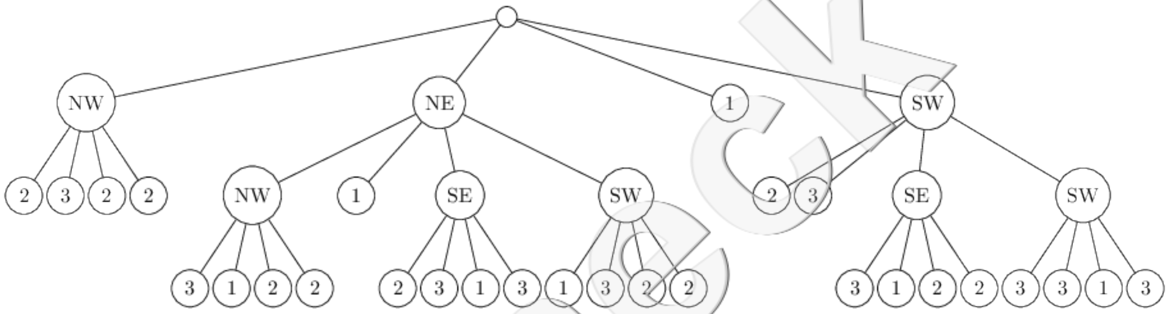
One each level, the search procedure will do at most $\Theta(n^{1/3})$ moves to the right. The number of levels is constant. Thus the worst-case complexity is $\Theta(n^{1/3})$.

**Problem 3**

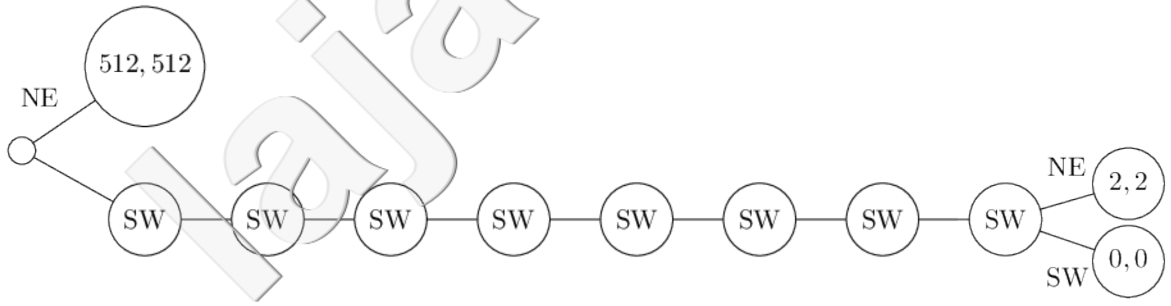
a) We accept both orientations NW, NE, SW, SE (Assignment)



and NW, NE, SE, SW (Lectures)



b) Following is one example of such points. The height is equal to $1 + \log_2(d_{\max}/d_{\min})$ where $d_{\max}$ is the size of the minimal bounding square of the points, and $d_{\min}$ is the size of the smallest sub-quadrant containing more than one point.



**Problem 4**

a) Programming question. See solution code attached.

b) The algorithm of a) performs a preprocessing which consists in a fast sorting algorithm, one time on the $x$-coordinate and one time on the $y$-coordinate. This takes $\Theta(n \log n)$. Then the algorithm splits the two sorted arrays in two with respect to the median. The split take time $\Theta(n)$. Finally it makes a recursive call of size $\lfloor n/2 \rfloor$ and one of size $(n - \lfloor n/2 \rfloor - 1) = \lceil n/2 \rceil - 1$. Hence the relation

$$T(n) \le T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil - 1) + O(n)$$

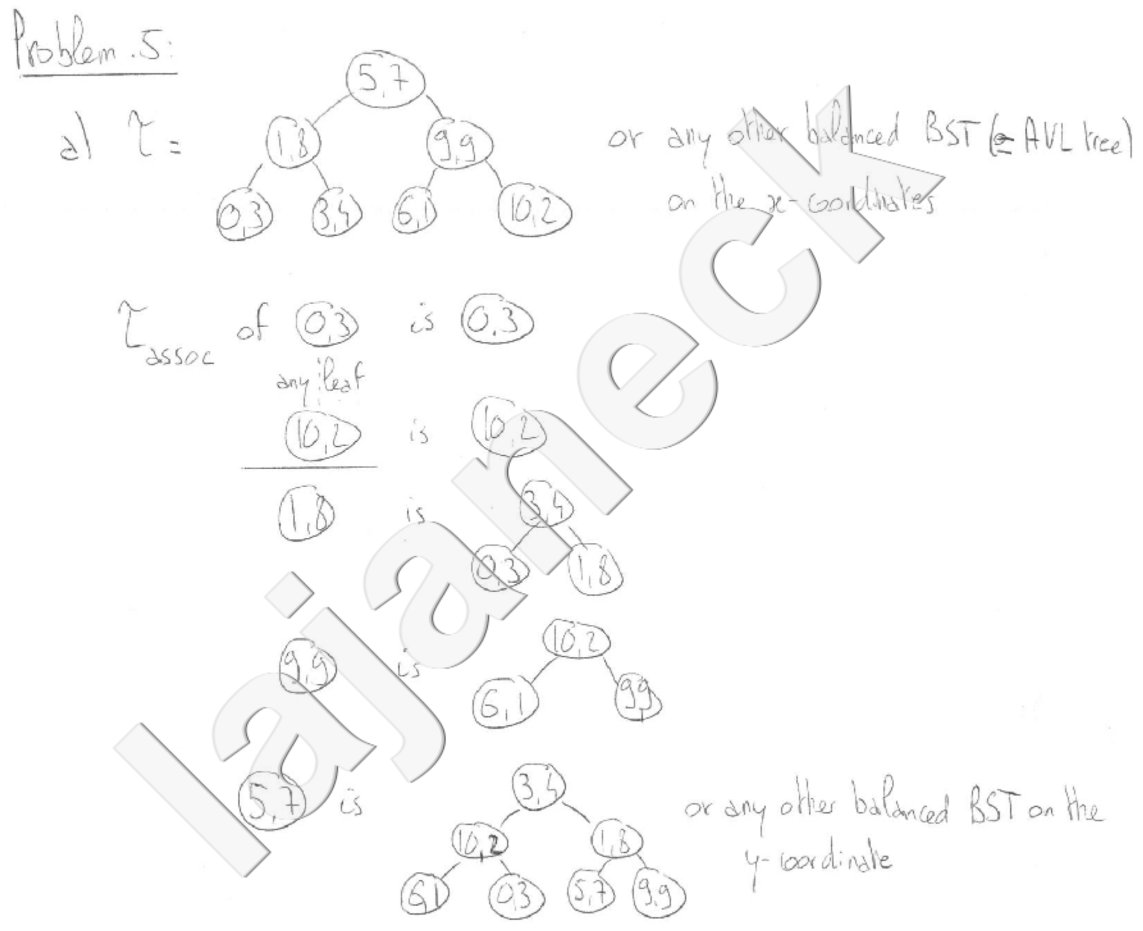
on the complexity of the computations. The associated sloppy recursion is

$$T(n) \leq 2\,T\,(n/2) + O(n).$$

with implies $T(n) \in O\,(n \log\,(n))$. Altogether, this yields a runtime $O(n \log n)$.

**Problem 5**

a)



b) Build a 1-dimensional range tree $\tau$ like in Slide 5 of Module 7. But store in every node an additional data: Store at node $v$ the number of nodes $\#P(v)$ in the sub-tree $P(v)$ of $\tau$ whose root is $v$. Now the range counting query is mostly similar to range query :

- for every outside node, do nothing
- for every boundary node, add one to the count if the node is in the range
- for every top inside node, add $\#P(v)$ to the count

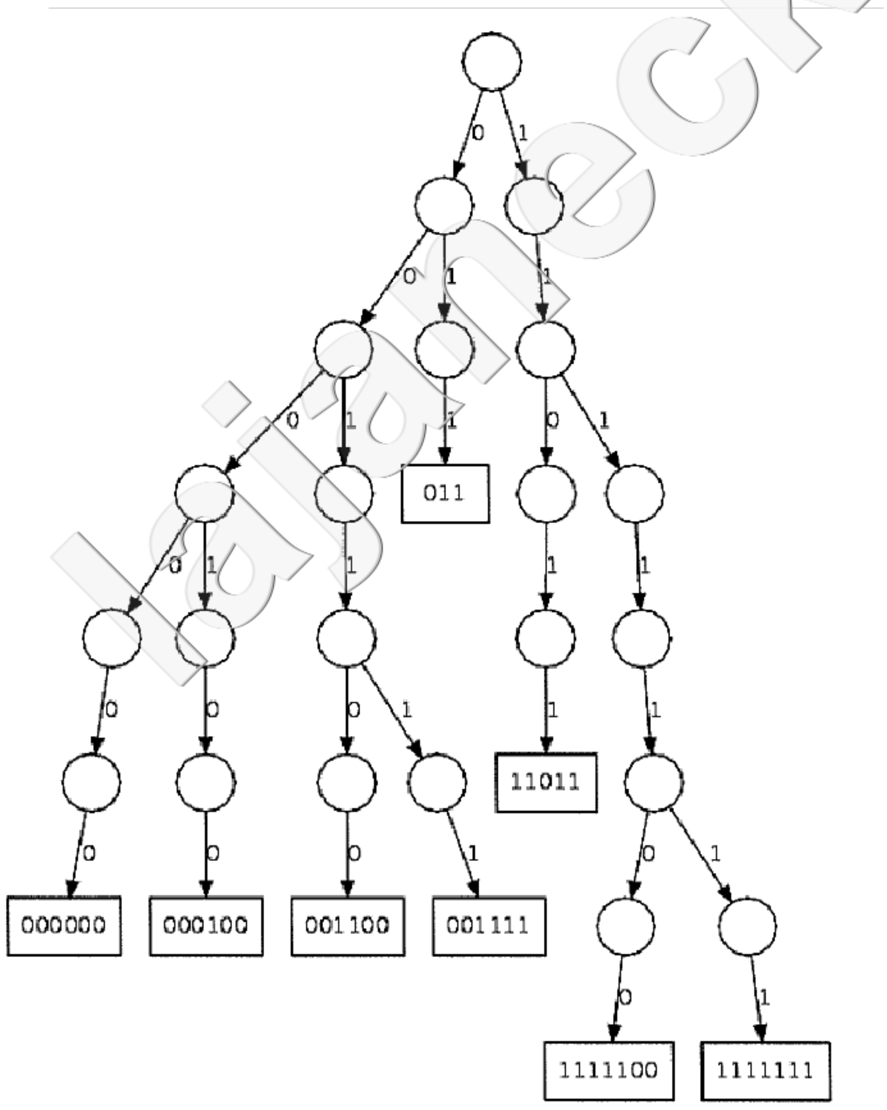
Cost is $\Theta(\log n)$ to compute the right and left boundaries and to treat them. Every top inside node takes constant time. Since there are $O(\log n)$ top inside nodes, we get a total worst-case complexity of $\Theta(\log n)$.

c)

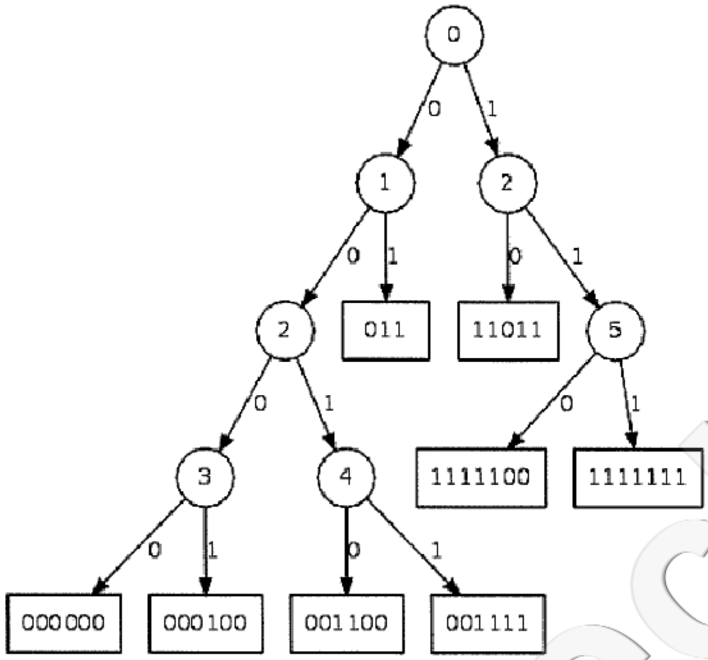
Build a 2-dimensional range tree as described in module 7, but use the modified tree from part (a) for the associated trees on the y-coordinates. For every "top" inside node v, perform a range counting query on the associated tree on the y-coordinate. Return the sum of the result of all the range counting queries on the associated trees, together with the number of boundary nodes that are within the region R. Since we need to perform boundary search on both x and y, so it's $O((\log n)^2)$.

**Problem 6**

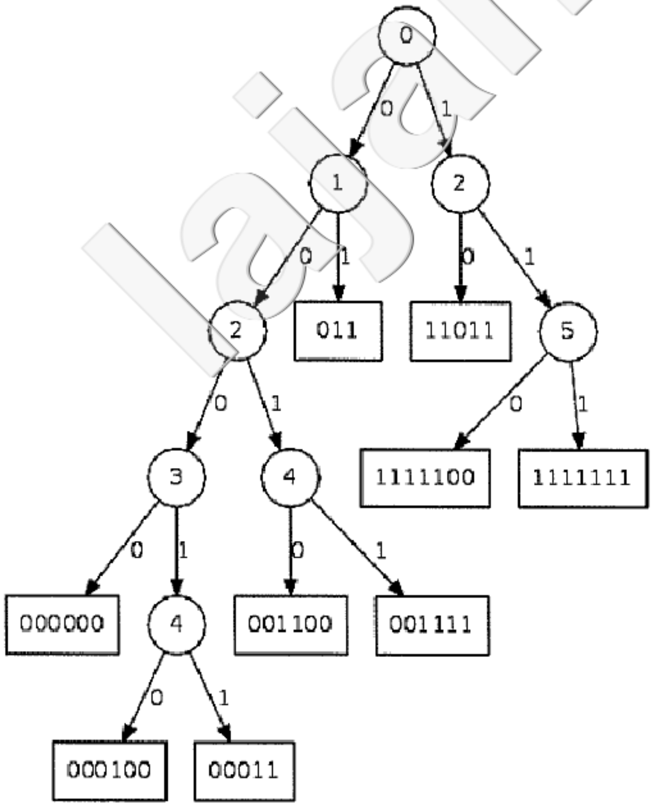
a)



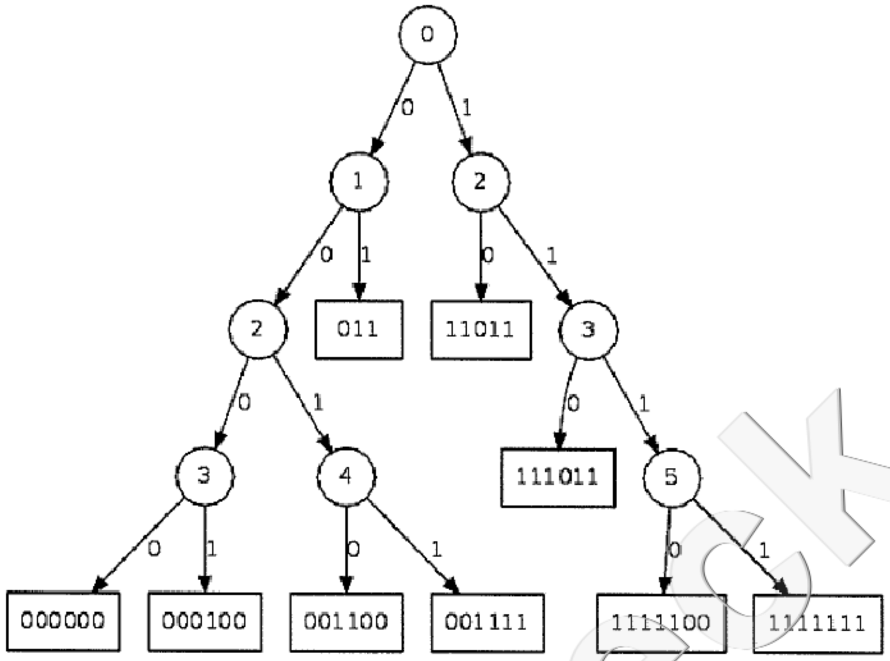
b)


0
1
2
011
11011
5
3
4
1111100
1111111
000000
000100
001100
001111

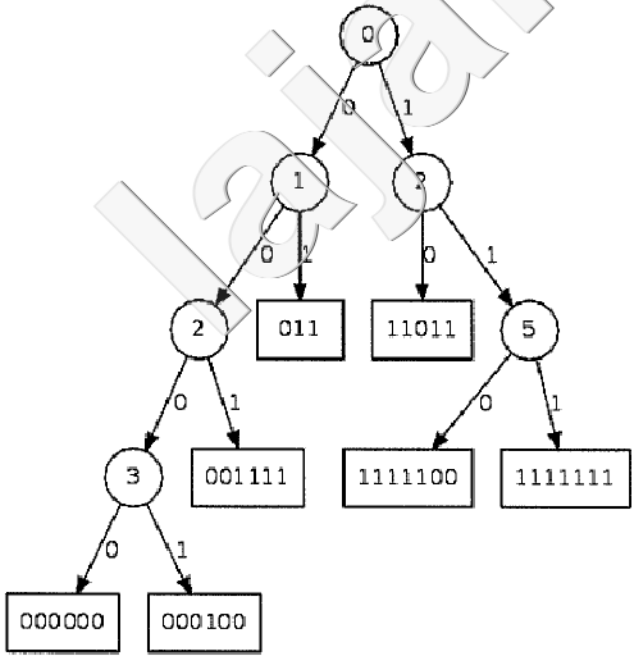
c)


0
1
2
011
11011
5
3
4
1111100
1111111
000000
4
001100
001111
000100
00011

d)


0
0
1
1
2
0
1
0
1
2
011
11011
3
0
1
0
1
3
4
111011
5
0
1
0
1
0
1
000000
000100
001100
001111
1111100
1111111

e)


0
0
1
1
2
0
1
0
1
2
011
11011
5
0
1
0
1
3
001111
1111100
1111111
0
1
000000
000100

**Problem 7**

a)

1. $P =$ she sells seashells

| $j$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $F[j]$ | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 2 | 3 | 0 | 0 | 1 |

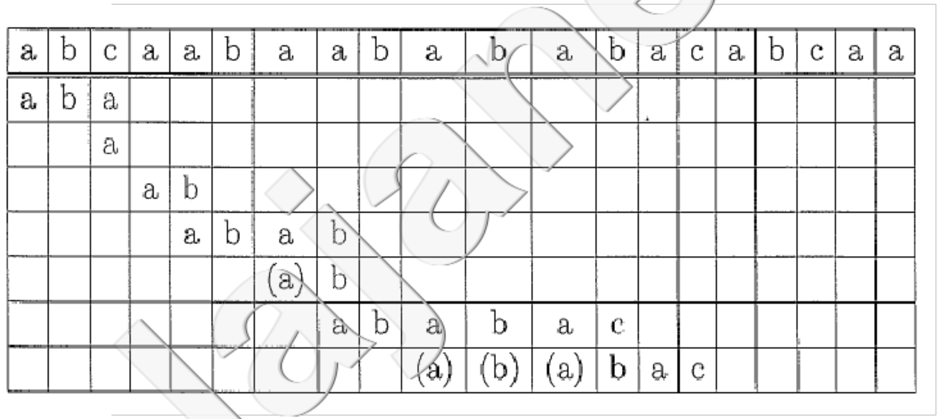
2. $P =$ abracadabracapabra

| $j$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $F[j]$ | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 0 | 1 | 2 | 3 | 4 |

3. $P =$ ababac

| $j$ | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| $F[j]$ | 0 | 0 | 1 | 2 | 3 | 0 |

b)



| a | b | c | a | a | b | a | a | b | a | b | a | b | a | c | a | b | c | a | a |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| a | b | a |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | a |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | a | b |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | a | b | a | b |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | (a) | b |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | a | b | a | b | a | c |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  | (a) | (b) | (a) | b | a | c |  |  |  |  |  |

**Problem 8**

a) For alphabet $\Sigma = \{a, b, c, d\}$, the last-occurrence function is

| $c$ | a | b | c | d |
|---|---|---|---|---|
| $L(c)$ | 4 | 5 | -1 | -1 |

. We accept any alphabet containing at least $a, b, d$ and the corresponding function.

b) For alphabet $\Sigma = \{a, b, c, d\}$, the suffix skip array is

| $i$ | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| $S[i]$ | $-3$ | $-2$ | $-1$ | $-3$ | $-2$ | 4 |

c)

Step 1)

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| a | a | a | b | a | b | d | a | a | b | a | a | a |
|  |  |  | a | a | b |  |  |  |  |  |  |  |

Mismatch at $T[3] = b$. $L[b] = 5$, $S[3] = -3$, so use bad suffix:
$i = i + m - 1 - min(L[T[i]], S[j]) = 3 + 6 - 1 - min(5, -3) = 11$

Step 2)

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| a | a | a | b | a | b | d | a | a | b | a | a | a |
|  |  |  | a | a | b |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  | b |  |

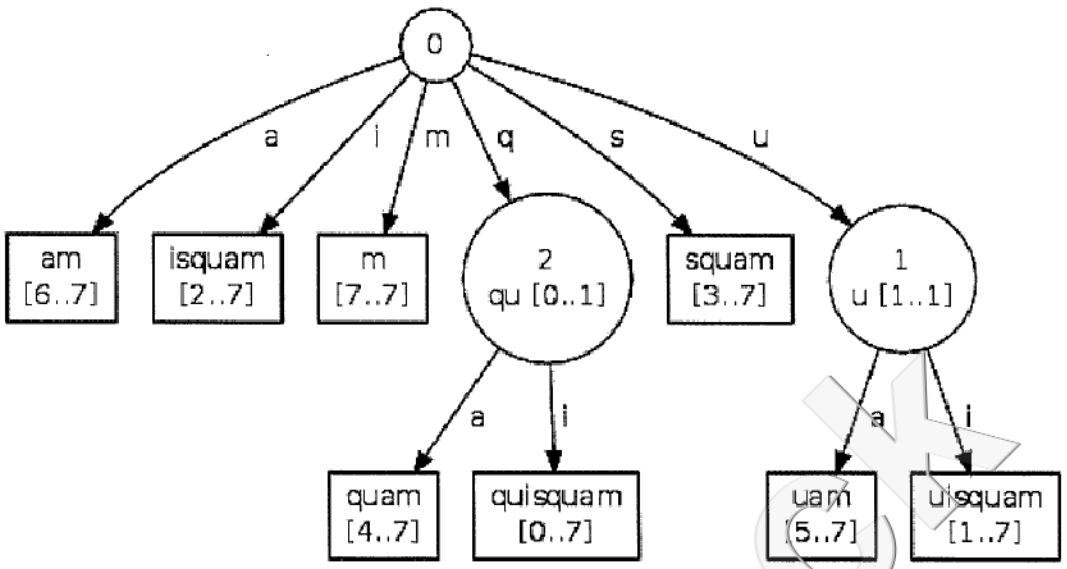
Mismatch at $T[11] = a$. $L[a] = 4$, $S[5] = 4$. Both bad character and good suffix give the same value:
$i = i + m - 1 - min(L[T[i]], S[j]) = 11 + 6 - 1 - min(4, 4) = 12$

Step 3)

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| a | a | a | b | a | b | d | a | a | b | a | a | a |
|  |  |  | a | a | b |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  | b |  |
|  |  |  |  |  |  |  |  |  |  |  |  | b |

Mismatch at $T[12] = a$. $L[a] = 4$, $S[5] = 4$.Both bad character and good suffix give the same value:
$i = i + m - 1 - min(L[T[i]], S[j]) = 12 + 6 - 1 - min(4, 4) = 13$
$i$ is no longer smaller than $n$, the length of the string.
Exit loop, no match was found.

**Problem 9**



**Problem 10**

a) [t|h|a|t|'|s|_|a|_|j|o|k|e|,|_|th|at|'s|_a|_j|ok|e,|_|s|o|n|,_|tha|t'|s_|a_|jo|ke]

b) 48 characters times 5 bits = 240 bits

c) 33 codes of 6 bits = 198 bits

d) Compression ration 165/240 ≃ 0.83. Considering the amount of repetition, the compression ratio is weak. This is because LZW is slow to build the frequent substrings.