(a)

P = adobodoa

Σ	a	b	c	d	o	t
L(c)	7	3	-1	5	6	-1

i	0	1	2	3	4	5	6	7
S[i]	-7	-6	-5	-4	-3	-2	-1	6

(b)

```
T = dotadotadotdotadobodoadot, n = 28
P = adobodoa, m = 8
i = j = (m - 1) = 7
```

Iteration 1

`T[7] = P[7]`

=> `i = 6, j = 6`

Iteration 2

`T[6] != P[6]`

=> `i = i + (m - 1) - min(L[T[i]], S[j]) = 6 + 7 - (-1) = 14`

`j = 7`

Iteration 3

`T[14] = P[7]`

=> `i = 13, j = 6`

Iteration 4

`T[13] != P[6]`

=> `i = i + (m - 1) - min(L[T[i]], S[j]) = 13 + 7 - (-1) = 21`

`j = 7`

Iteration 5...12

`T[21..14] = P[7..0]`

=> `i = 13, j = -1`

Return match `i + 1 = 14`

(c)

```
boyer-moore-all(T, P)
Takes text T and pattern P
Reports all match indices
Returns FAIL if there were no reports
 1. L ← last occurrence array computed from P
 2. S ← good suffix array computed from P
 3. i ← m − 1, j ← m − 1
 4. while i < n do
 5.     if T[i] = P[j] then
 6.         i ← i − 1
 7.         j ← j − 1
 8.     else
 9.         i ← i + m − 1 − min(L[T[i]], S[j])
10.         j ← m − 1
11.     if j = -1 then
12.         report i + 1    // match at index i+1
12.         i ← i + m + 1
13.         j ← m − 1
14. if report.size() = 0 then
15.     return FAIL
```

(d)

Let `T = axyzb` and `P = xyz`. Now, `n = 5`, `m = 3`, `i = j = (m - 1) = 2`

Iteration 1

Condition `T[2] = P[2] or T[1] = P[1]` fails.

`i = i + (m - 1) = 2 + 2 = 4` and `j = 2`

Iteration 2

Condition `T[4] = P[2] or T[3] = P[1]` fails.

`i = i + (m - 1) = 4 + 2 = 6` and `j = 2`

As `i > n`, we found no matches using the heuristic while P is a substring of T.

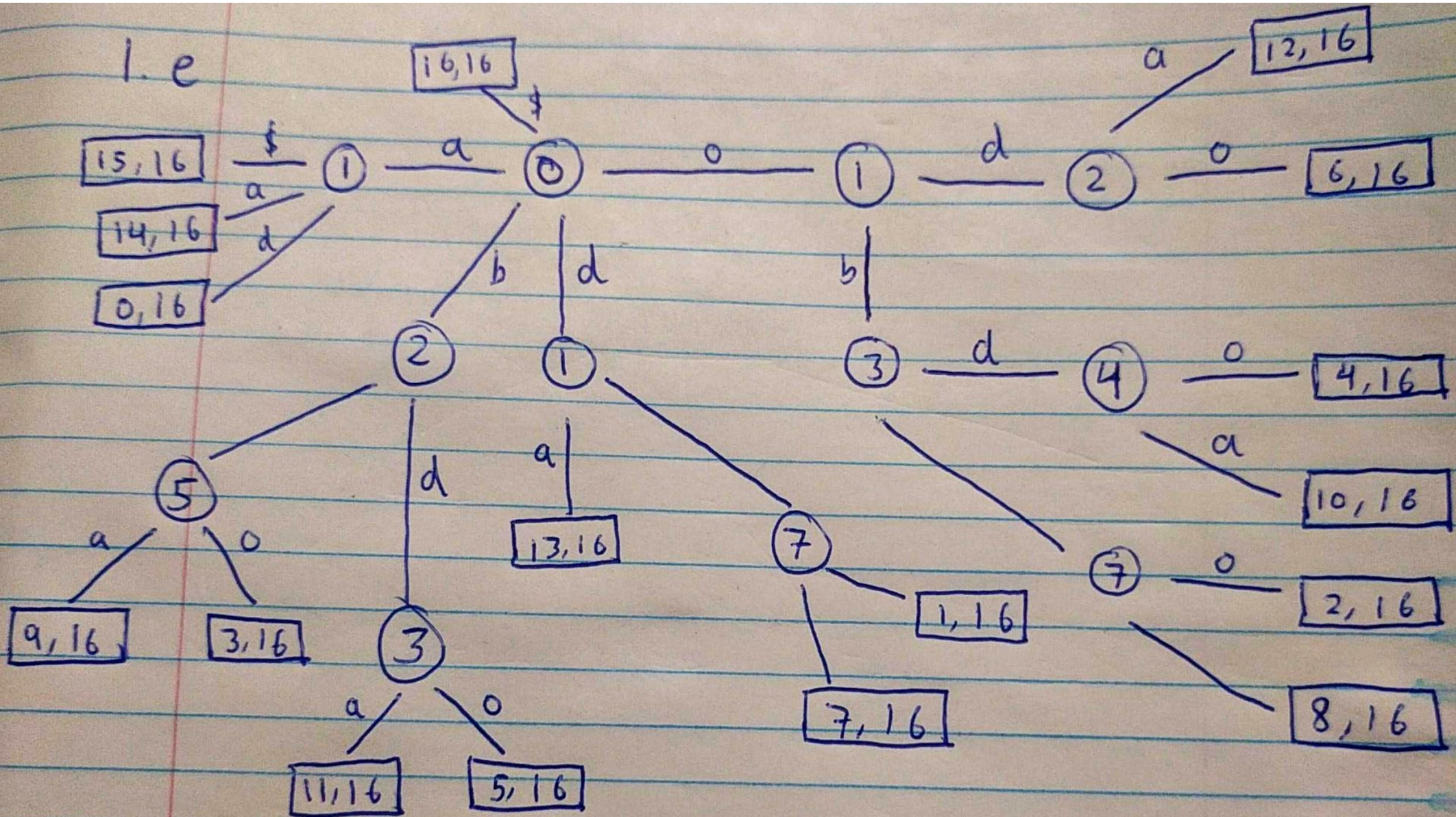
(f)

```
M = 17
h(k) = k % M
T, n = 7
P = 181, m = 3
h(P) = 181 % 17 = 11
```

We find a T of length 7 that would result in the full substring being checked at every iteration. For that to happen, the hash of the substring must `= 11`.

We choose `T = 555555555`. We see for any substring of length 3, we get 11 using the hash function. This means the algorithm would check every character of the substring with every counterpart character in P. However `555 != 181`. Therefore it would result in `θ(nm)` matches, worst-case for Rabin-Karp Fingerprint algorithm.

1. e


16,16
$
15,16
$
a
14,16
d
0,16
a
o
b
d
a
o
12,16
6,16
4,16
10,16
2,16
8,16
1,16
7,16
13,16
9,16
3,16
11,16
5,16