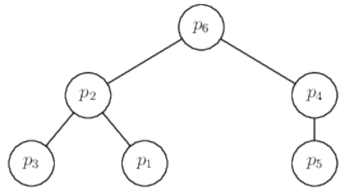
1. Let $p_1 = (30, 45)$, $p_2 = (5, 20)$, $p_3 = (10, 14)$, $p_4 = (80, 80)$, $p_5 = (50, 30)$, $p_6 = (35, 40)$. The kd-tree is shown below:



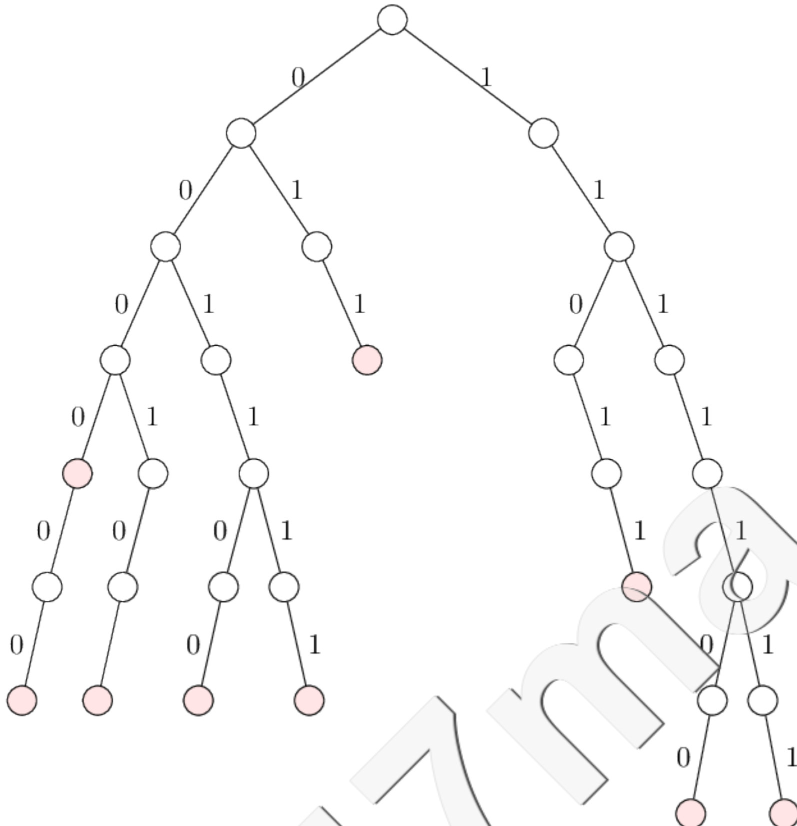
2. part (a): Let $\max_{ij}$ denote the maximal array value in the interval $[i, j]$ and let $\min_{ij}$ denote the minimal array value on the interval $[i, j]$; that is, $\max_{ij} = \max\{ A[k] \mid i \leq k \leq j \}$ and $\min_{ij} = \min\{ A[k] \mid i \leq k \leq j \}$. Then $D_{ij} = \max_{ij} - \min_{ij}$. We can find both $\min_{ij}$ and $\max_{ij}$ for any interval $[i, j]$ using range trees. We represent every entry $A[i] = v$ as a point (i, v). All points are stored in a balanced tree T according to their first coordinates. In every node u we keep the highest value m_u in the subtree of u. That is, m_u is the largest v, such that (i, v) is stored in the node u or one of its descendants. In every node u we also keep the smallest value l_u in the subtree of u. That is, l_u is the smallest v, such that (i, v) is stored in the node u or one of its descendants.

   A query $[i, j]$ is answered as follows. For any range (i, j), let P_1 denote the search path for i in the range tree T and let P_2 denote the search path for j in T. We define boundary nodes and inside nodes in the same way as in Module 7 p. 19. Then $\max_{ij}$ is the maximum value among (i) all m_w stored in a top inside node w and (ii) all v such that (k, v) is stored in a boundary node and $i \leq k \leq j$. Analogously $\min_{ij}$ is the minimum value among (i) all l_w stored in a top inside node w and (ii) all v such that (k, v) is stored in a boundary node and $i \leq k \leq j$. We must compare $O(\log n)$ values to find $\min_{ij}$ and $\max_{ij}$. Therefore a query is answered in $O(\log n)$ time. Since we keep only two additional values in every node, the tree T needs $O(n)$ space.

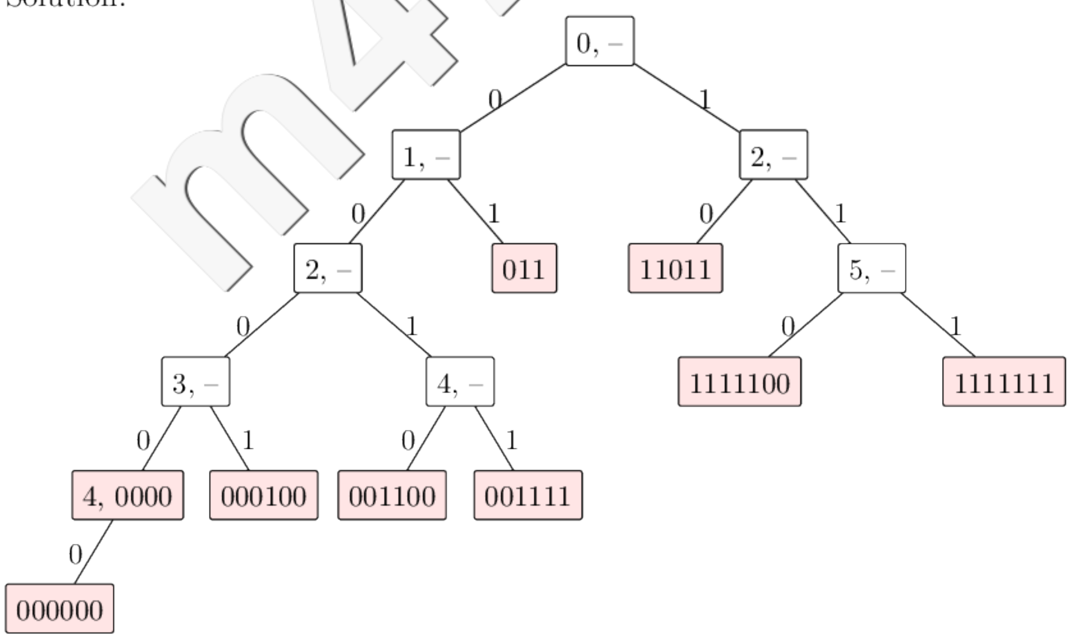
   part (b): The key observation is that $Q = [x_1, x_2] \times [0, h]$ contains at least one point if and only if the point p_m is in Q, where p_m is the lowest point with x-coordinate in $[x_1, x_2]$. We can find p_m using the same method that was used in part (a) to find $\min_{ij}$. All points are stored in the tree according to their x-coordinates. In every node u we keep l_v, where l_v is the lowest y-coordinate of a point stored in v or one of its descendants. We also keep a point $p_v = (x_v, l_v)$ with y-coordinate l_v in the node v.

   To find p_m, we find the minimum among (i) all l_w stored in a top inside node w and (ii) all v such that (k, v) is stored in a boundary node and $x_1 \leq k \leq x_2$. Then p_m is the corresponding point with minimum y-coordinate. If the y-coordinate of p_m does not exceed h, we report p_m. Otherwise there are no points in $[x_1, x_2] \times [0, h]$ and the answer is `NO`.

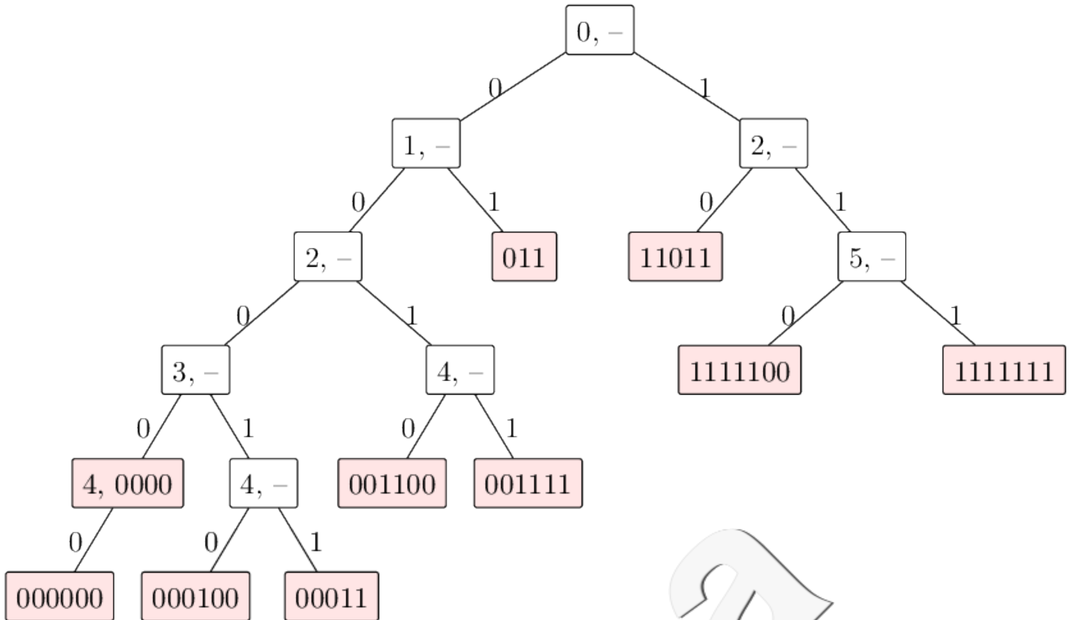
3. (a) Solution:



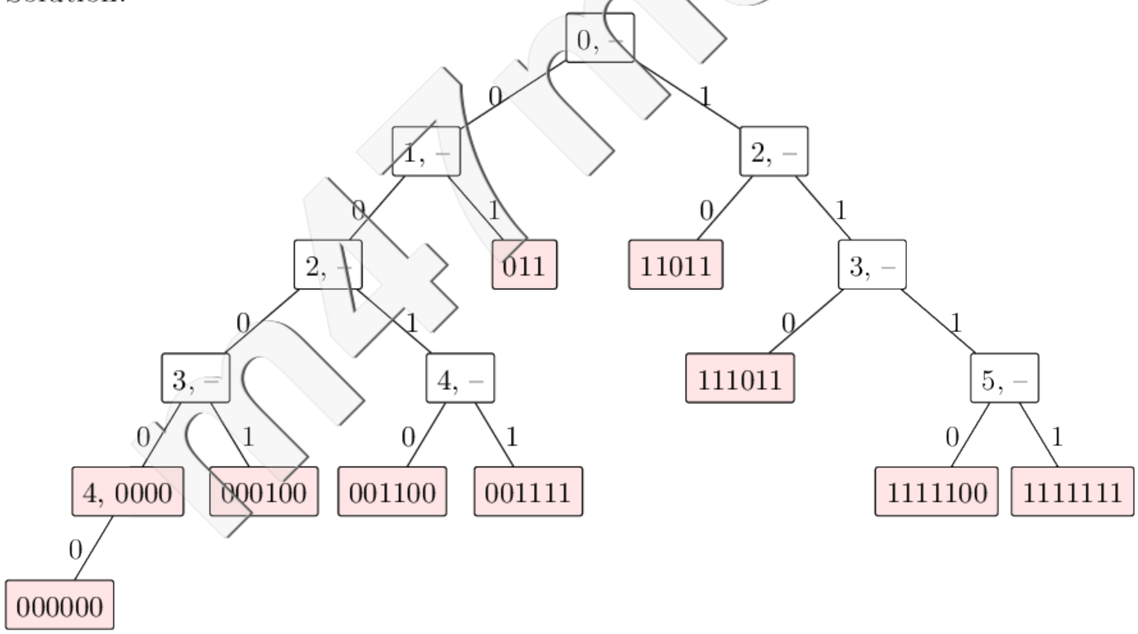
(b) Solution:



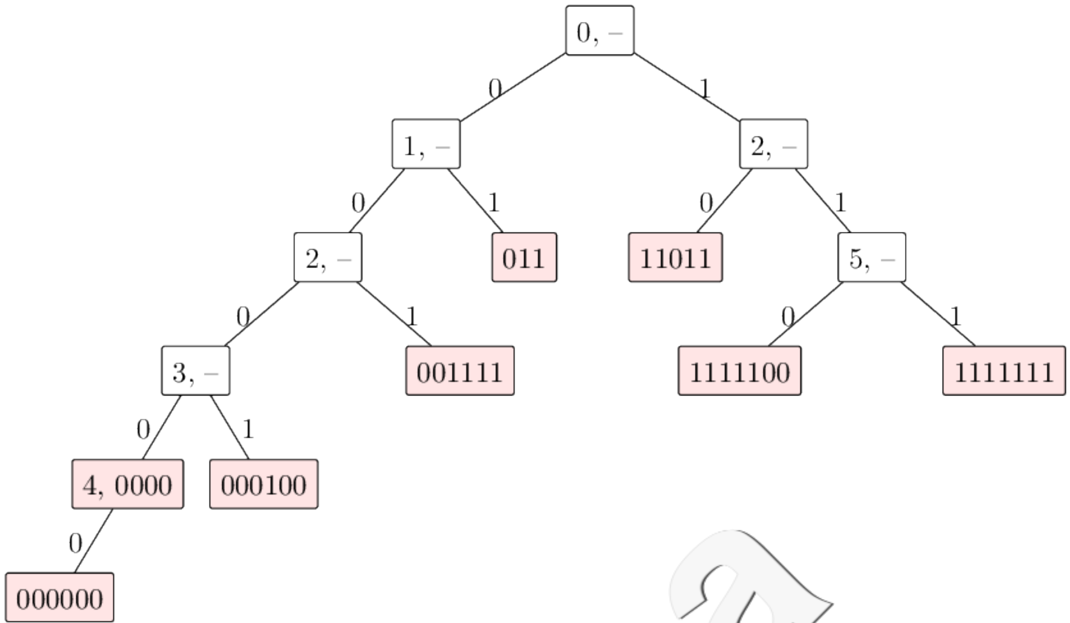
(c) Solution:



(d) Solution:



(e) Solution:



4. (a) i. F = 0 0 0 0 1 0 0 0 1 0 1 0 0 1 2 3 0 0 1
   ii. F = 0 0 0 1 0 1 0 1 2 3 4 5 6 0 1 2 3 4
   iii. F = 0 0 1 2 3 0

(b)
```
T =   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
```

(c) Let PT denote the concatenation of P with T, and let F be the KMP failure function for PT.
Observe that the sequence $S_j = F[j], F[F[j]-1], F[F[F[j]-1]-1], \ldots, 0$ enumerates lengths of all proper suffixes of $PT[0 \ldots j]$ that are also prefixes of $PT[0 \ldots j]$. Note that an empty string is both a proper prefix and a proper suffix of a non-empty string.

We wish to find the first j with $j \geq 2m-1$ with the property that $m \in S_j$. If this property holds for some j, then there is an m-length suffix of $PT[0 \ldots j]$ that is also an m-length prefix of $PT[0 \ldots j]$, which is just P itself. This observations leads to the following recipe:

Search for the minimal $j \geq 2m-1$ such that either:

- $F[j] = m$, or
- $F[j] > m$, and $F[F[j]-1] = m$ or $F[F[F[j]-1]-1] = m$, etc.

and return $i = j - 2m + 1$. If no such j exists then P does not occur in T.

Example 1: Consider P = aca, and T =abcaaca. The failure function for PT is

$$F[0 \ldots 9] = 0, 0, 1, 1, 0, 0, 1, 1, 2, 3$$

The recipe returns $j = 9$. Indeed, the first occurrence of P in T is at i = j - 2m + 1 = 4.

Example 2: Consider P = aca, and T = caca. The failure function for PT is:

$$F[0 \ldots 6] = 0, 0, 1, 2, 3, 4, 5$$

The recipe returns $j = 6$. Indeed, the first occurrence of P in T is at i = 6 - 2m + 1 = 1.

5. (a)

a	b	c	d
4	5	-1	-1

(b) S = -3 -2 -1 -3 -2 4

(c) We start with $i = j = 5$, match "ab" successfully, mismatch at "b" with $i = j = 3$. The bad character rule returns 5; the good suffix rule returns -3. We move i to $3+5-(-3) = 11$, j moves to 5. This forces a mismatch at "b" with $i = 11; j = 5$. the bad character rule returns 4, good suffix rule returns 4. Advance i to 12, $j = 5$. We again mismatch at "b", and then the pattern shifts off the end of the text.

6. (a) Yes, you can construct such a quad tree in $O(nlogn)$ time, in order to see this notice that time required to construct a quadtree is $O(nh)$, where h is the height of the quadtree. Since we know from class that the height of a quadtree is $O(log(\frac{d_{max}}{d_{min}}))$, and every point must be at an integer coordinate, it follows that the height of our quadtree is $O(\sqrt{2}n^2)$. Thus the time to construct our quadtree is $O(nlog(\sqrt{2}n^2)) = O(nlog(\sqrt{2}) + nlog(n)) = O(nlogn)$.