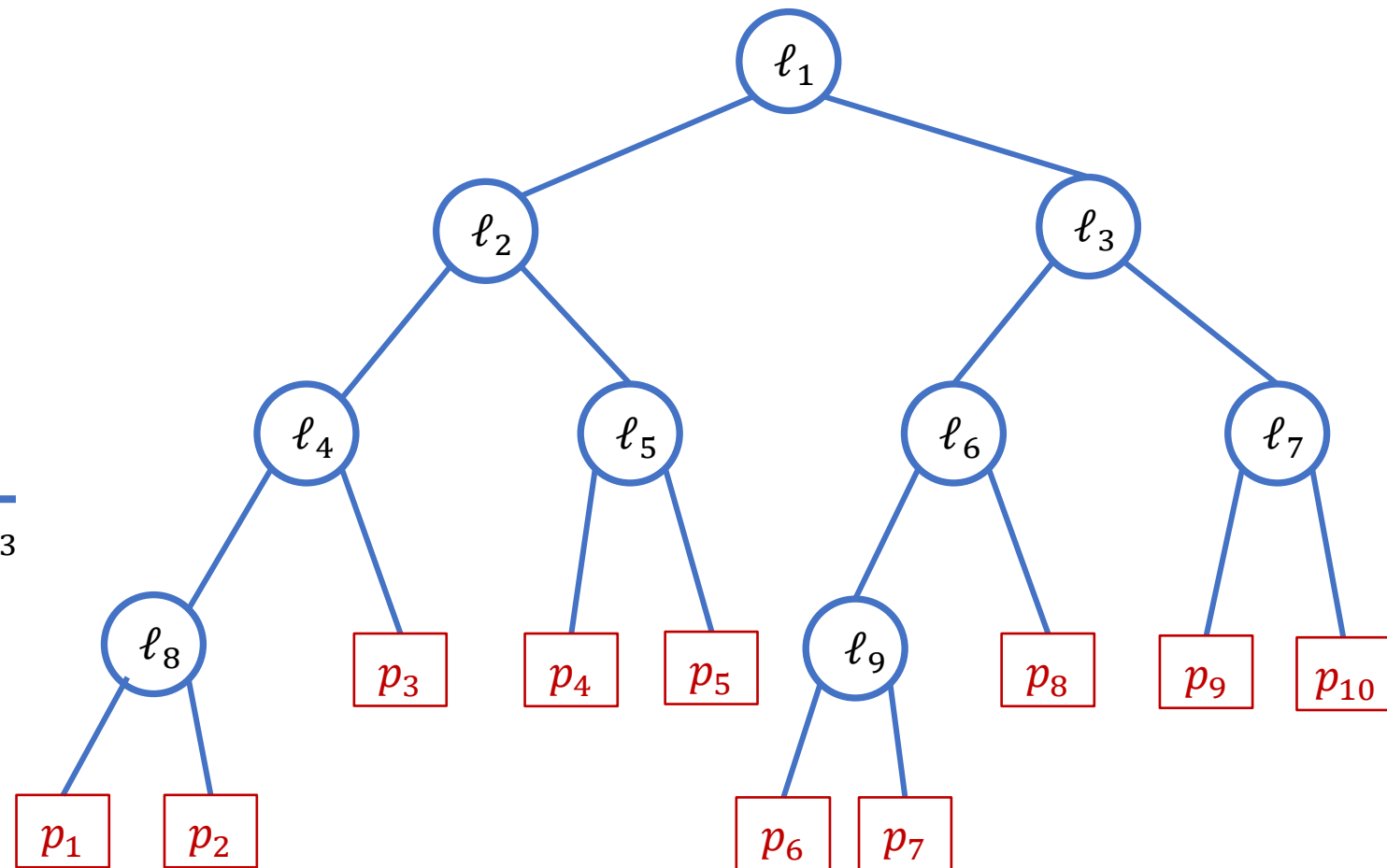
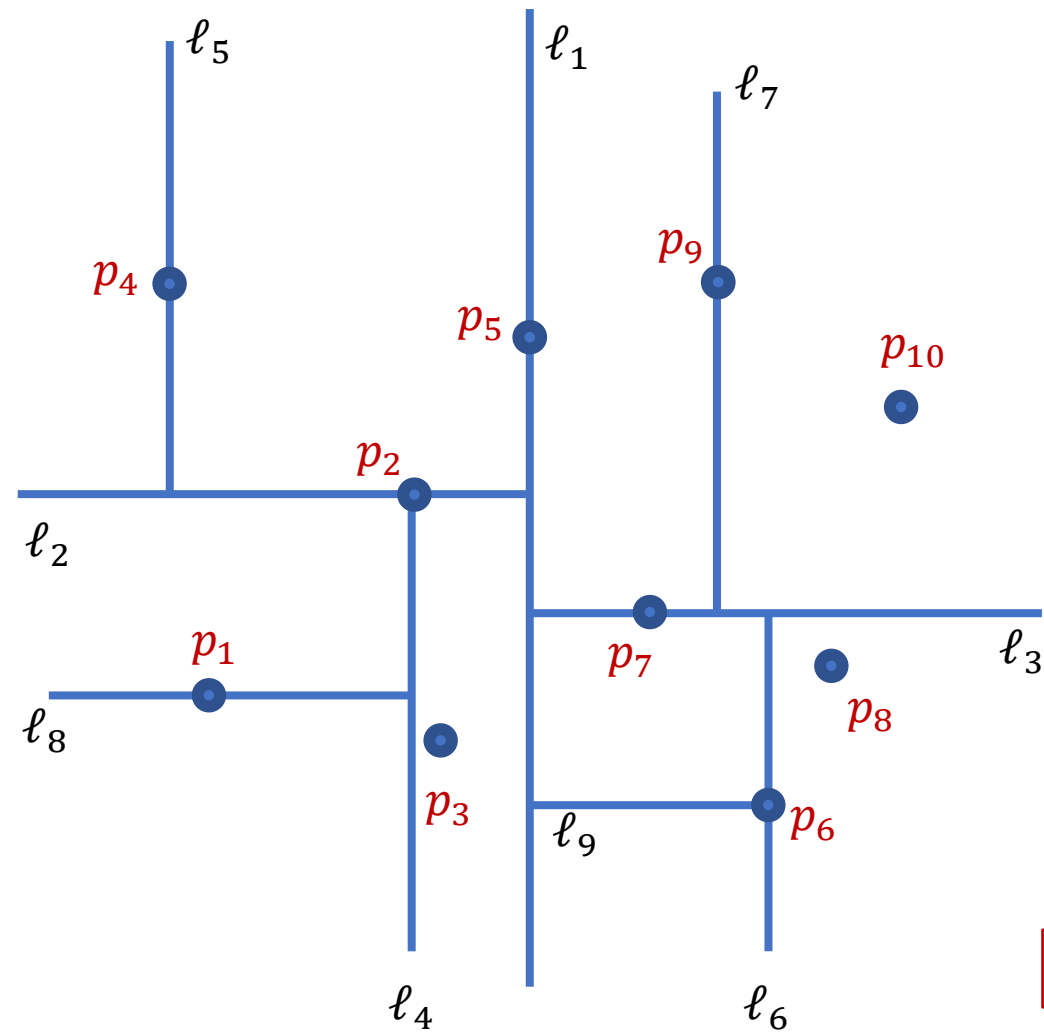


# Nearest-neighbor search using Kd-trees

# Kd-tree



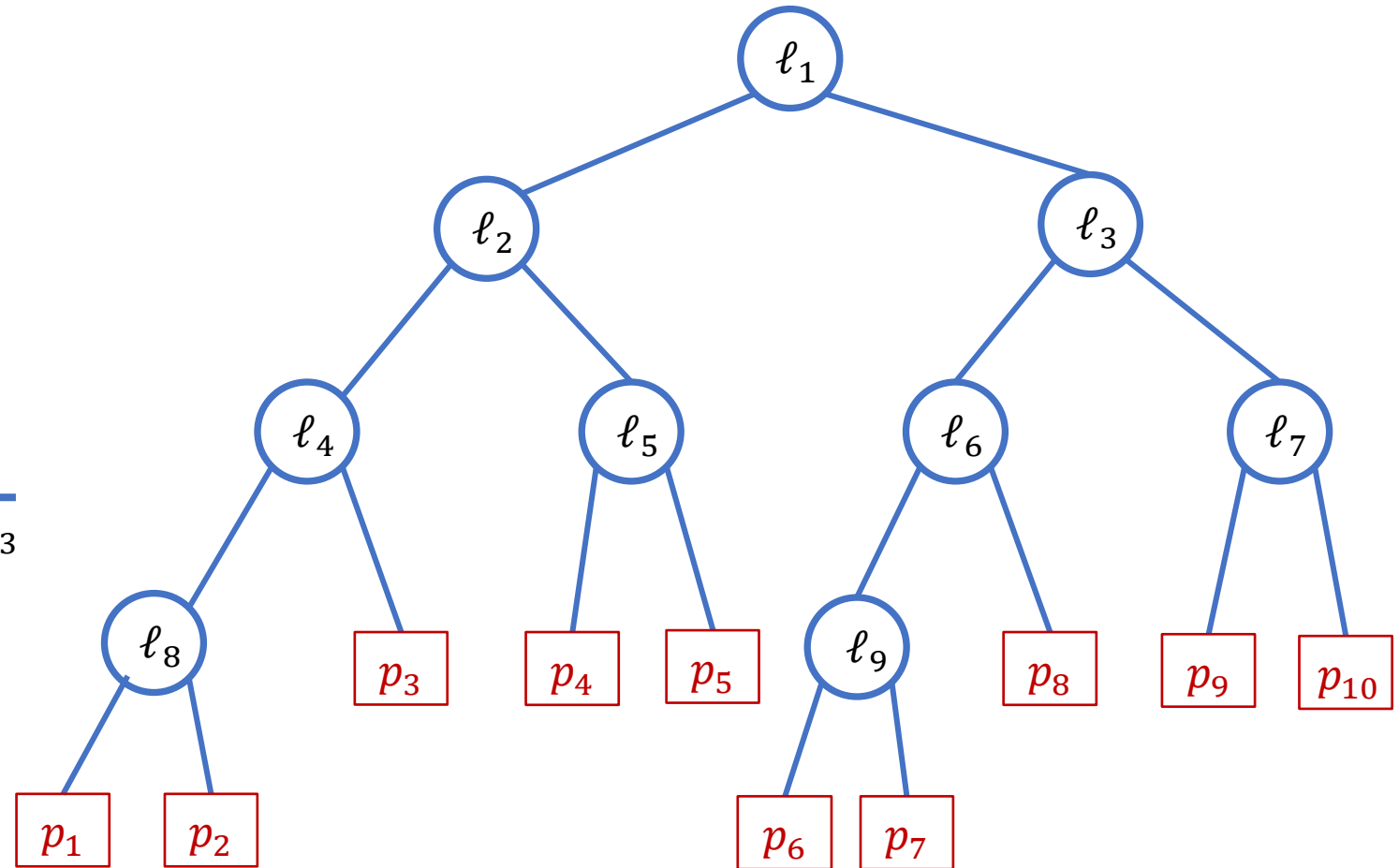
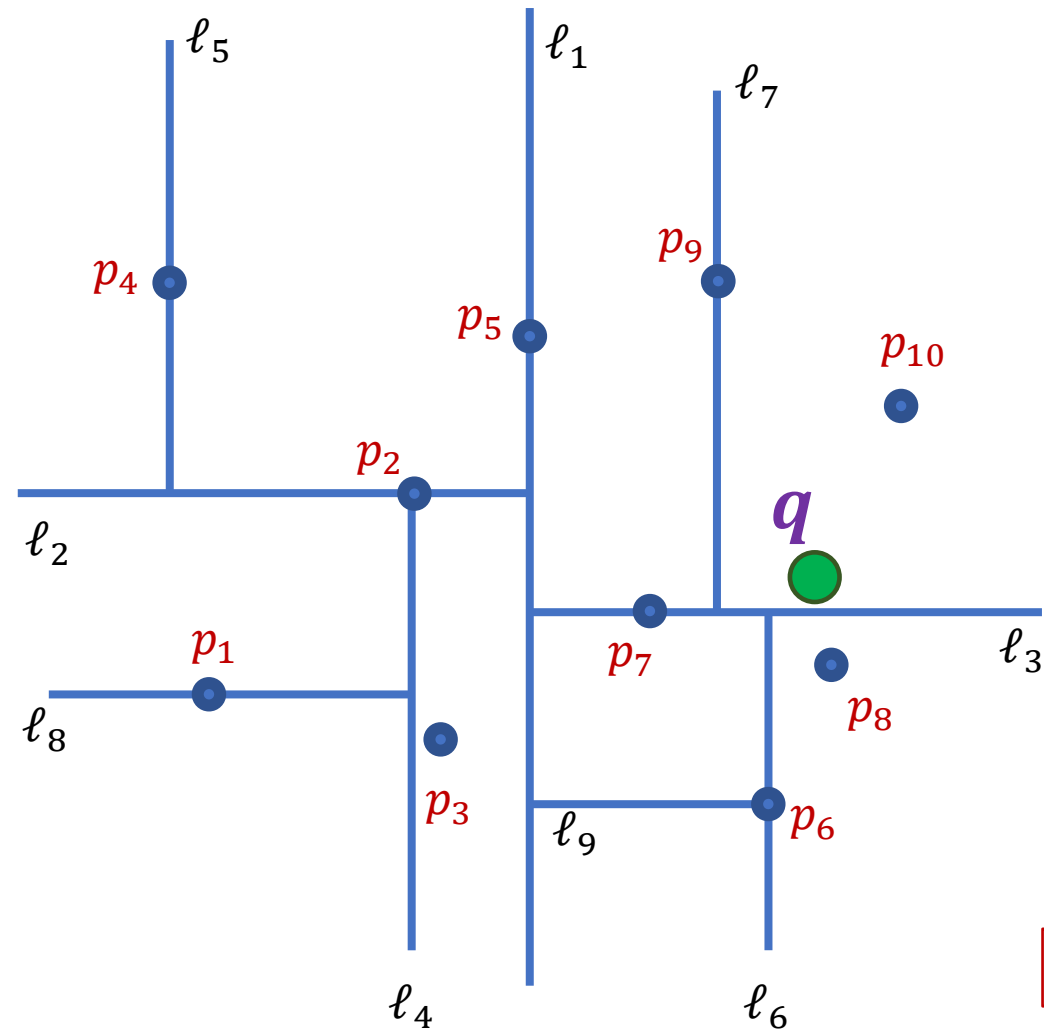
# The nearest-neighbor problem

- Given a set of points  $P$  and a query point  $q$ , return the point  $p \in P$  closest to  $q$
- Often, we would like to preprocess  $P$  into a data structure that efficiently answers such queries

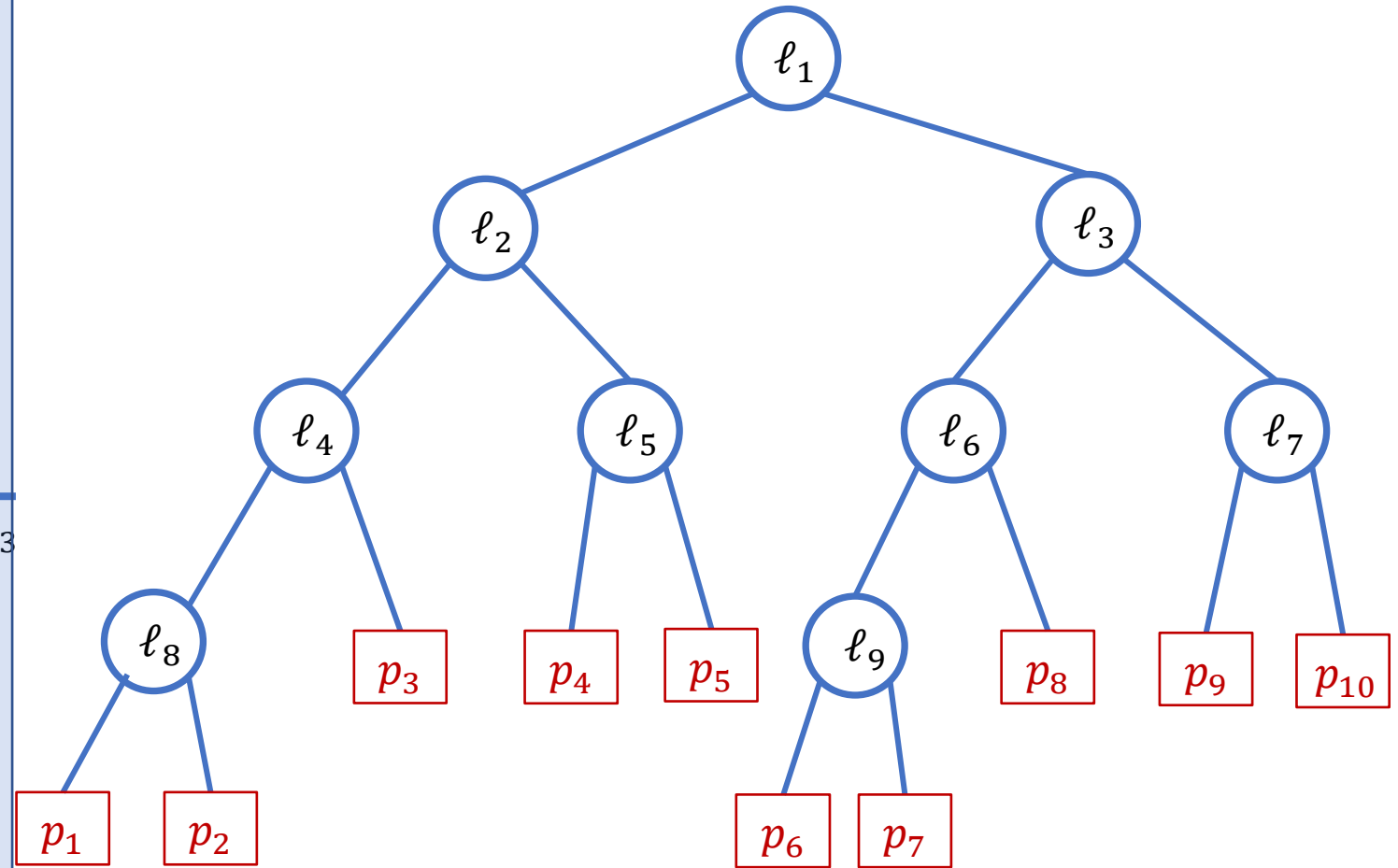
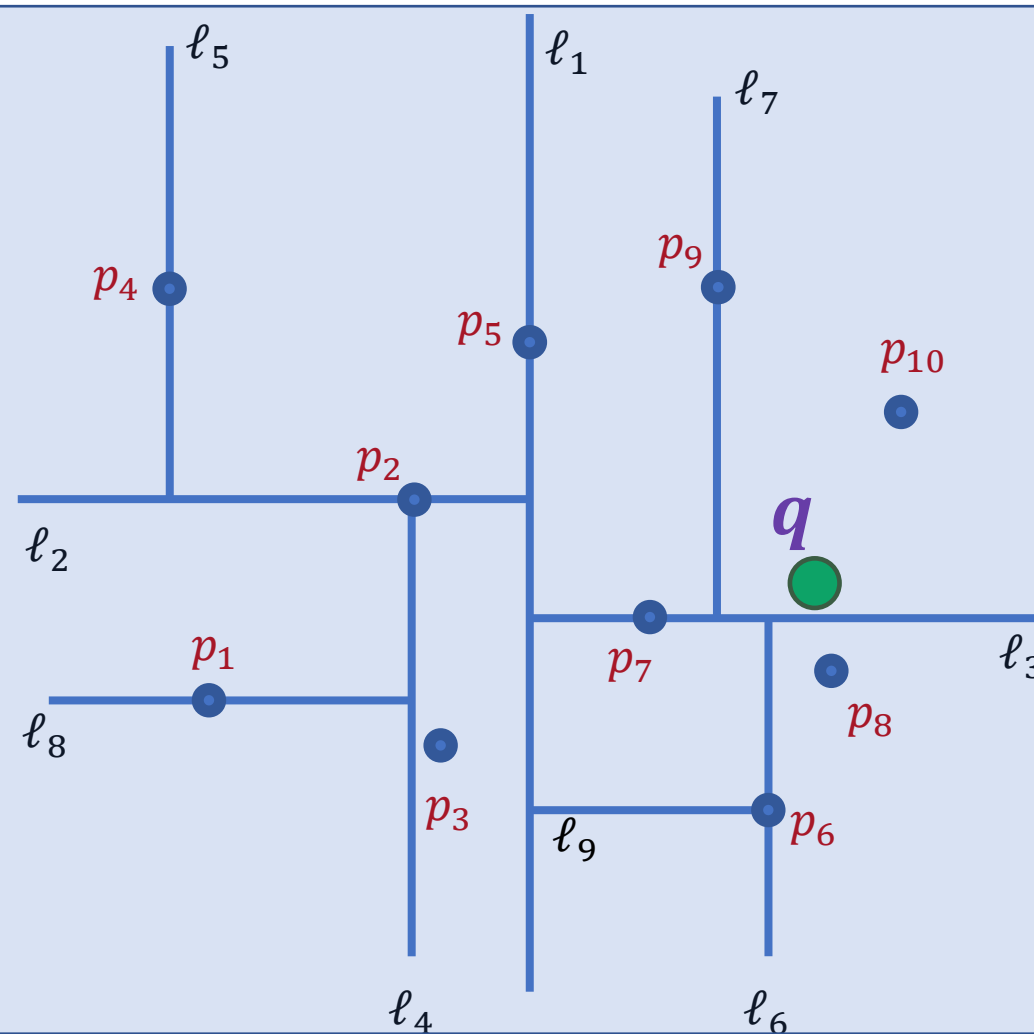
# Nearest-neighbor search using a Kd-tree

- Search the tree recursively to find the point (leaf) in the same cell as the query point  $q$
- This is our candidate for the nearest neighbor of  $q$
- On the way up search each subtree where a closer point than the current candidate might be found

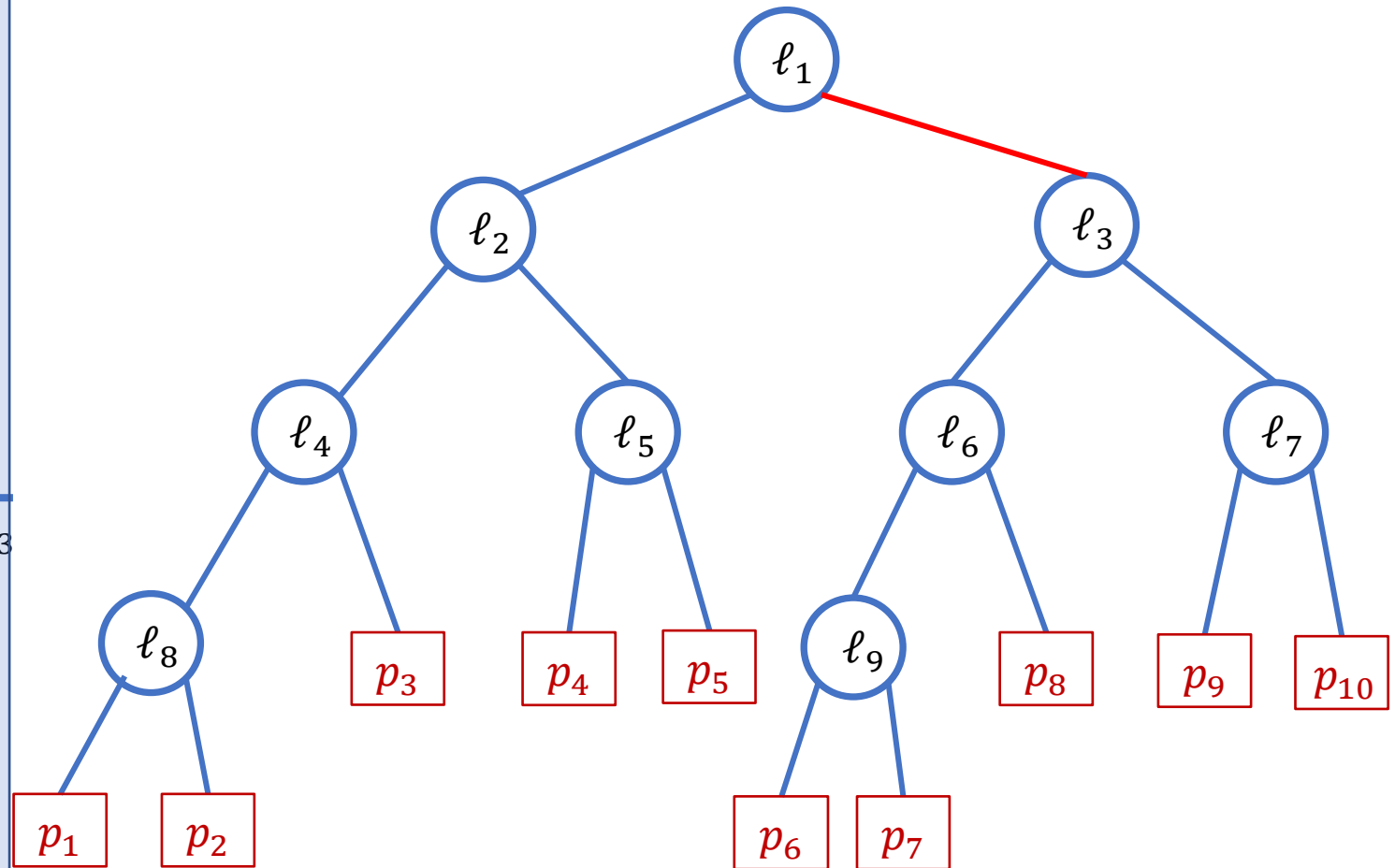
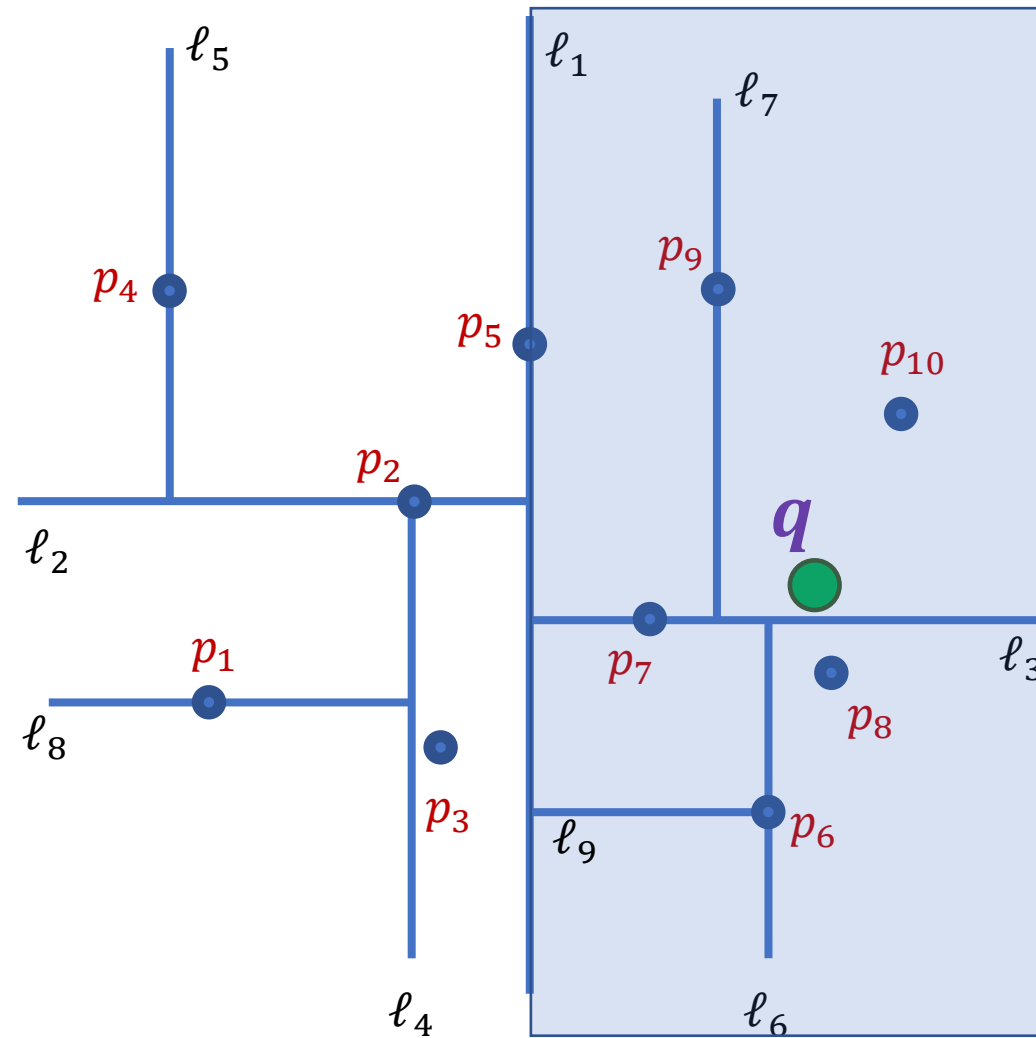
# Nearest-neighbor search using Kd-tree



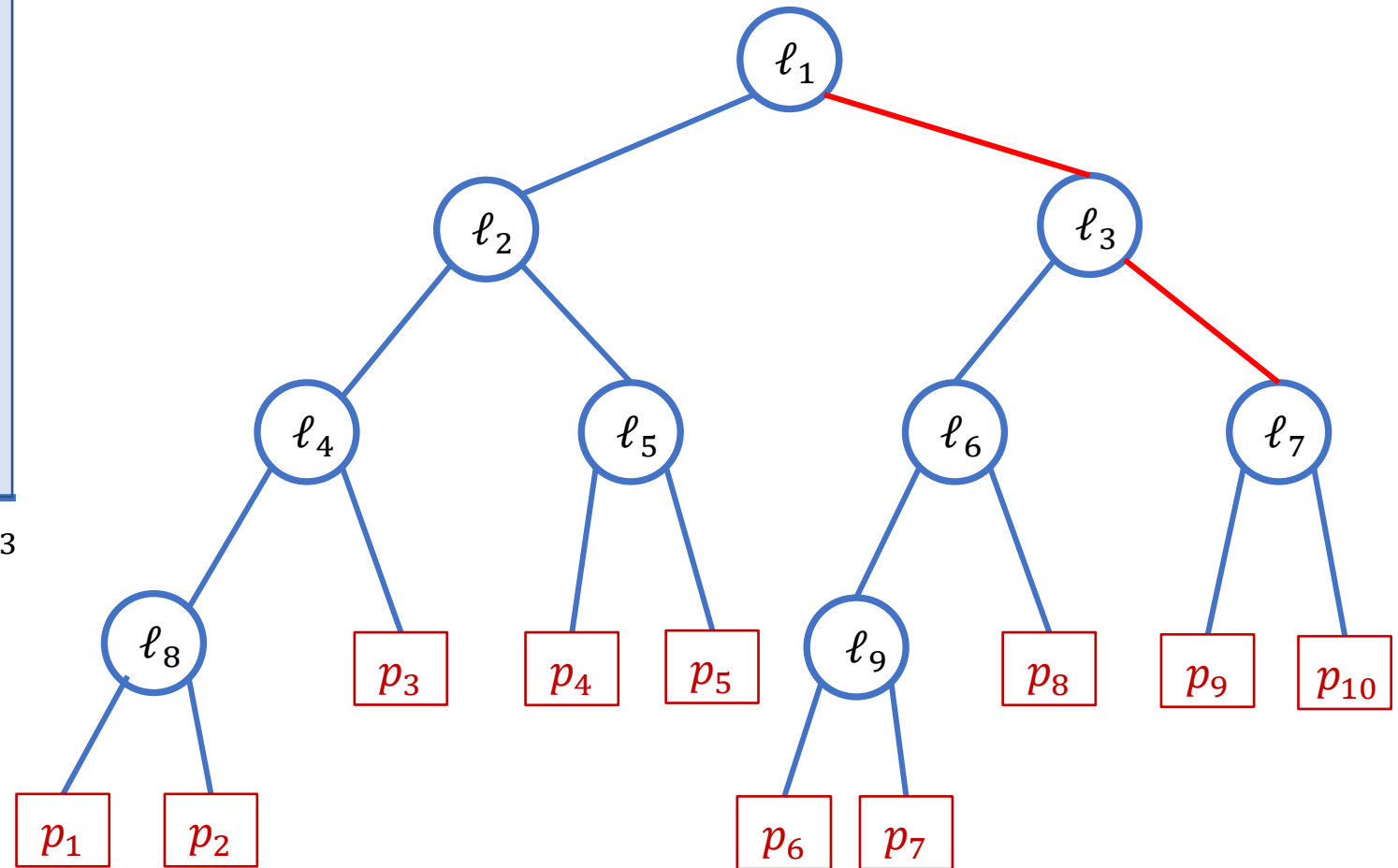
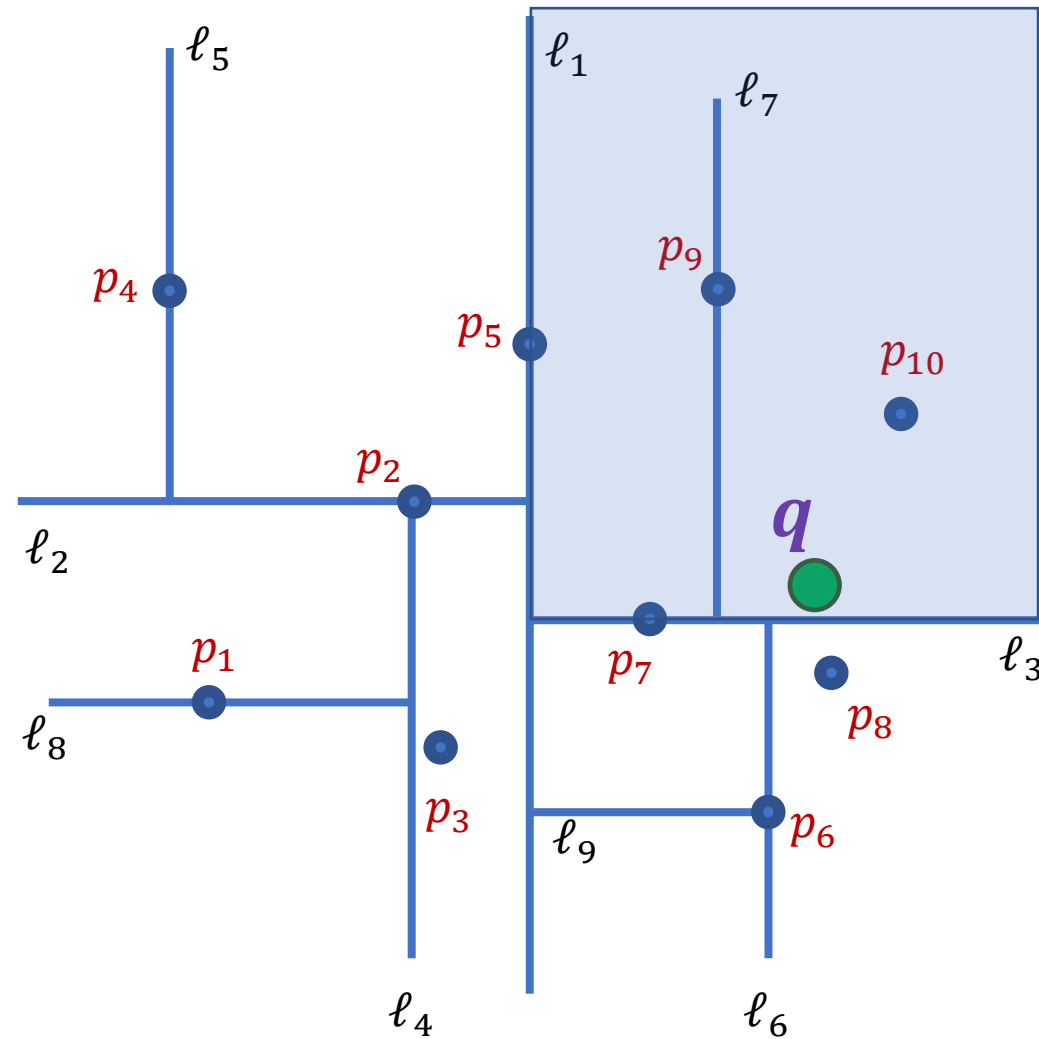
# Step 1: find the cell of $q$



# Step 1: find the cell of $q$

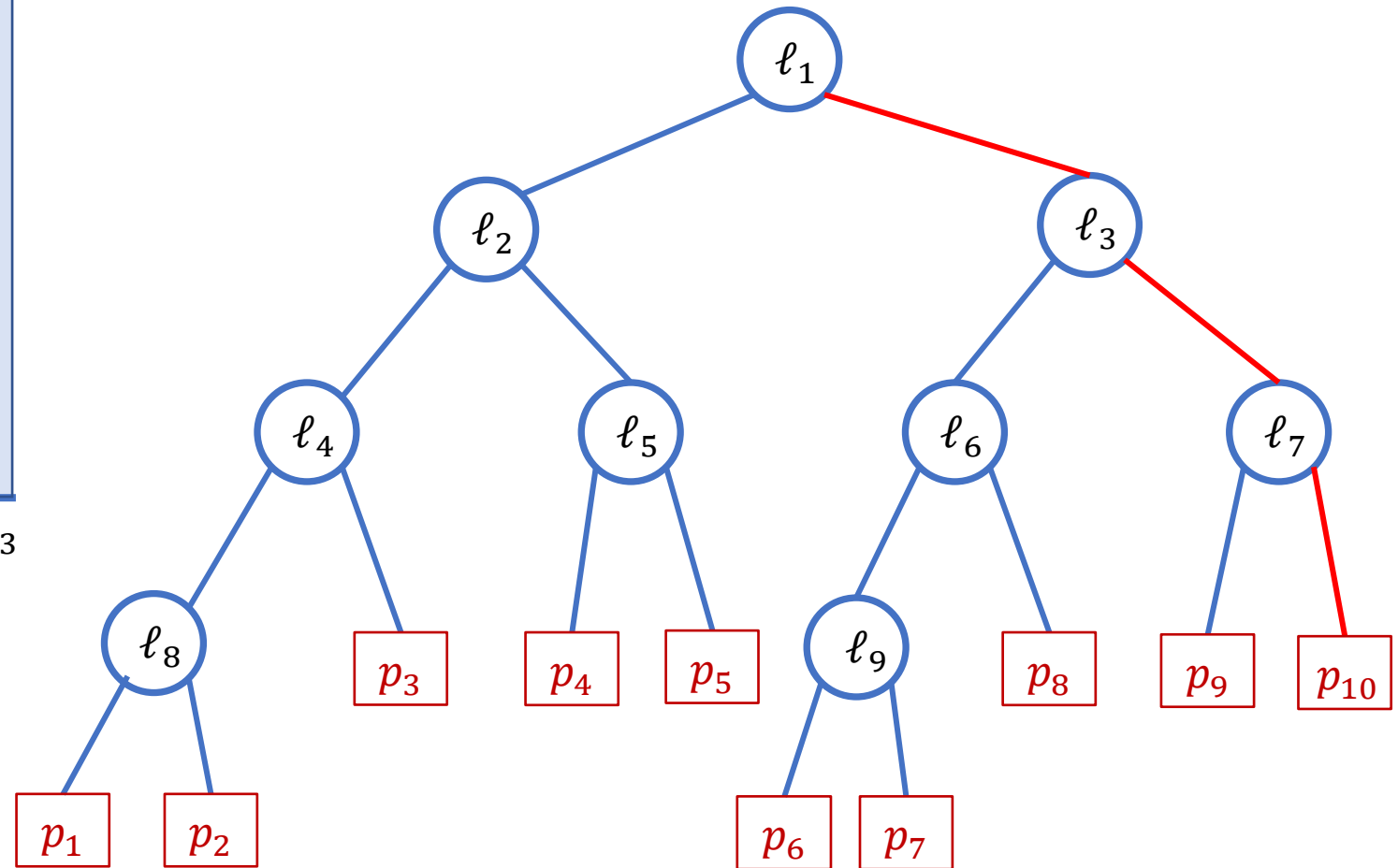
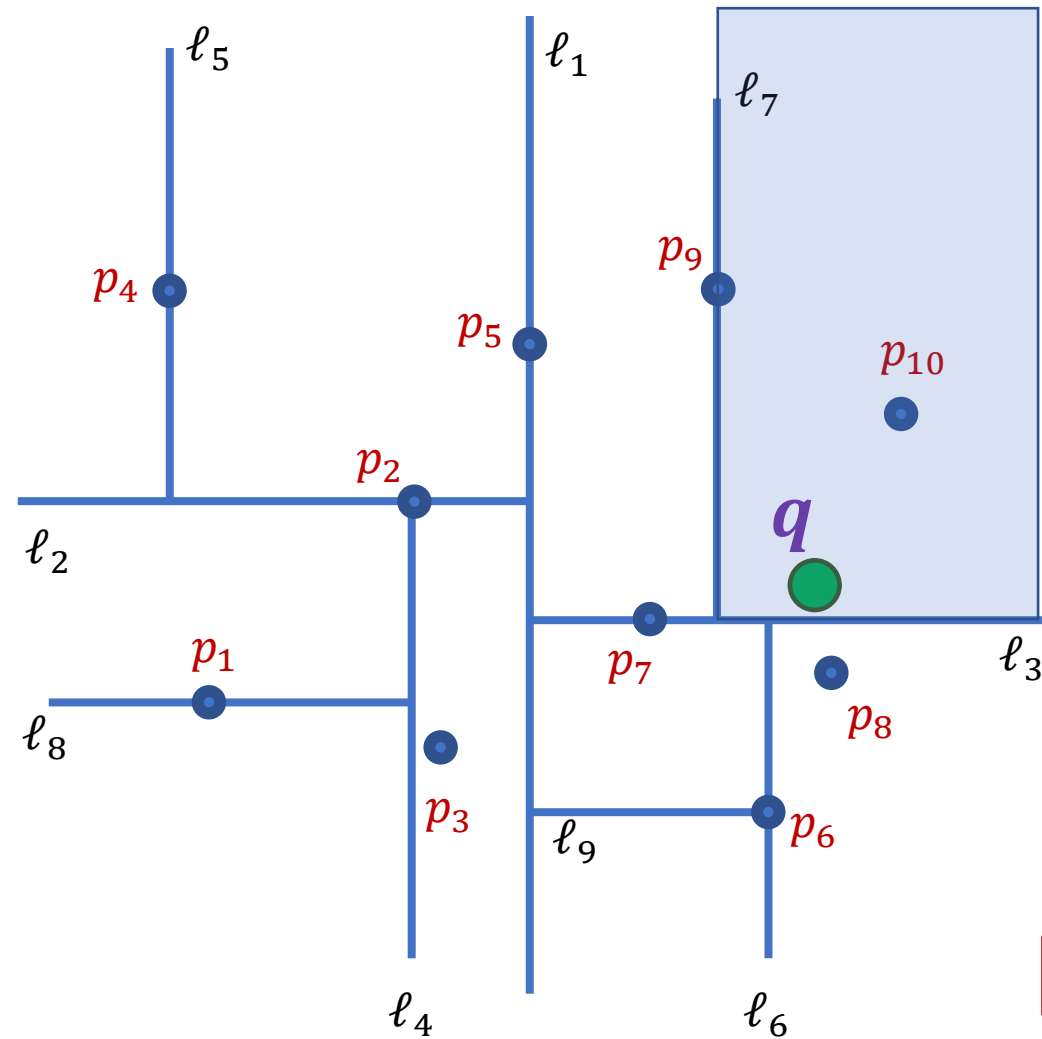


# Step 1: find the cell of $q$

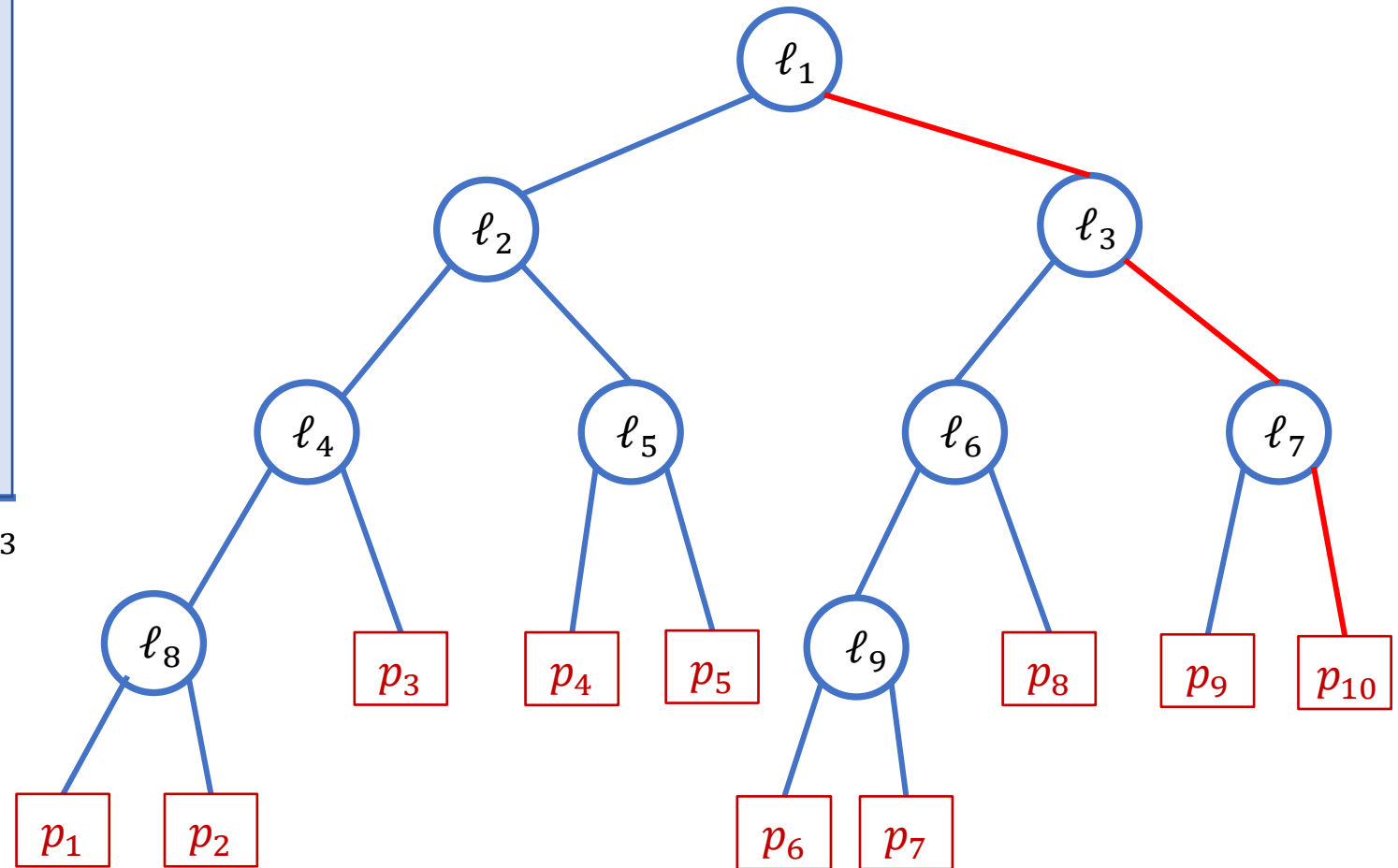
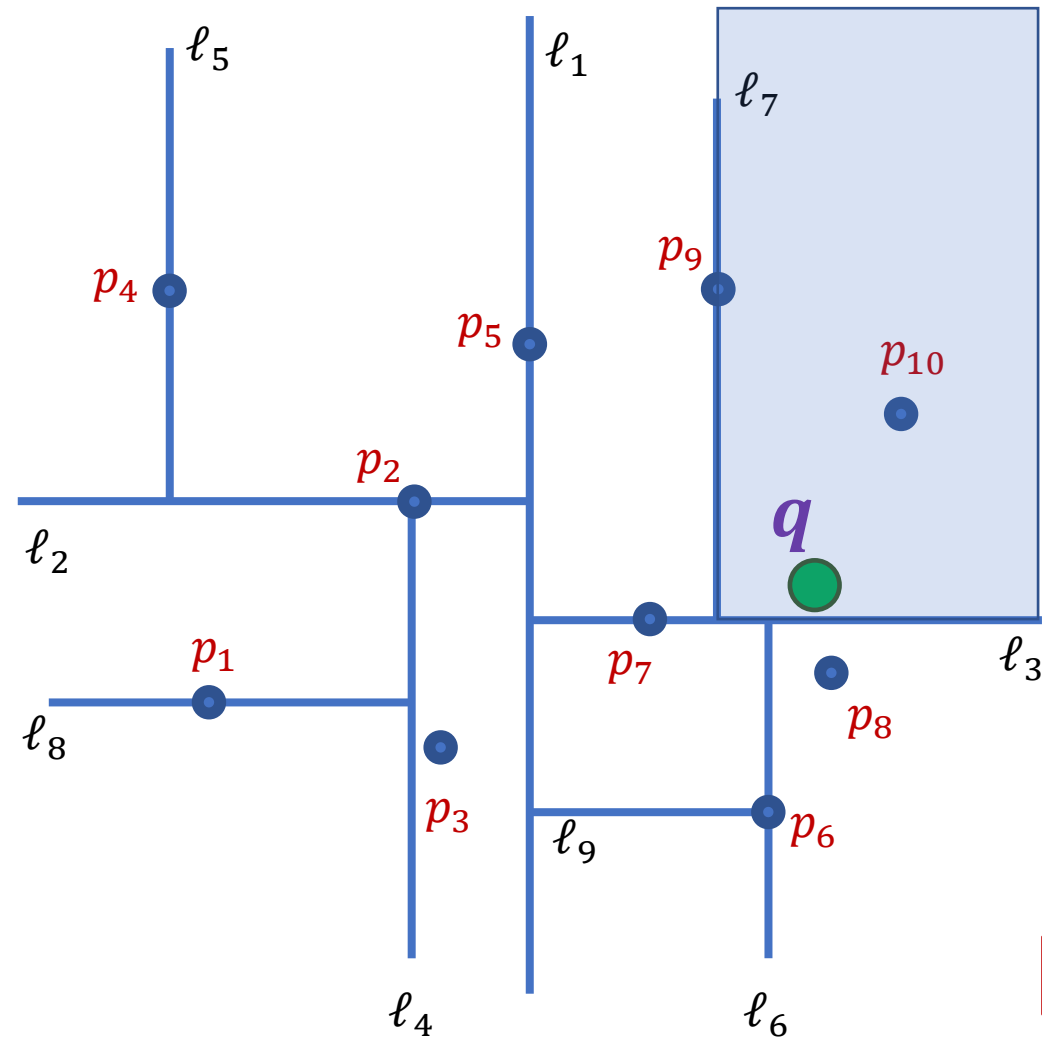




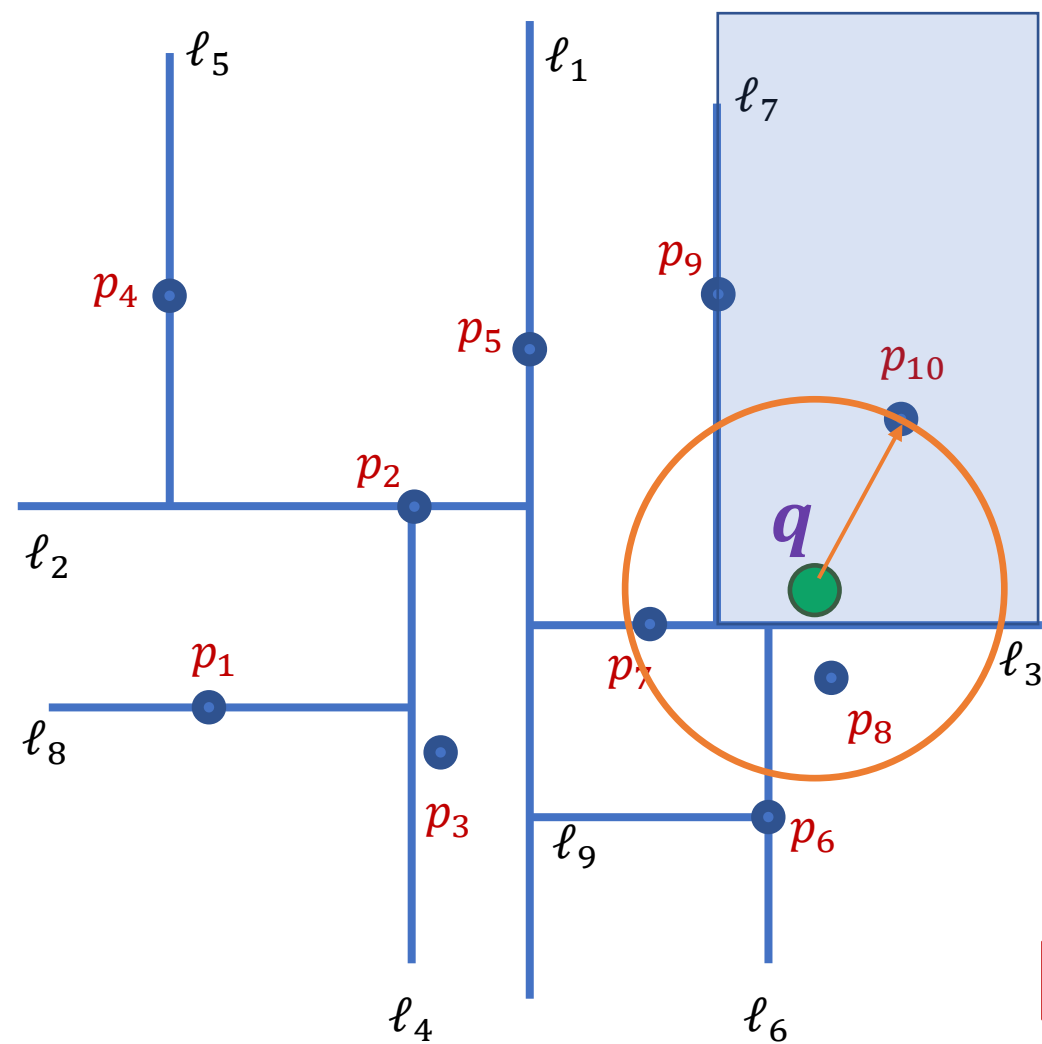
# Step 1: find the cell of $q$



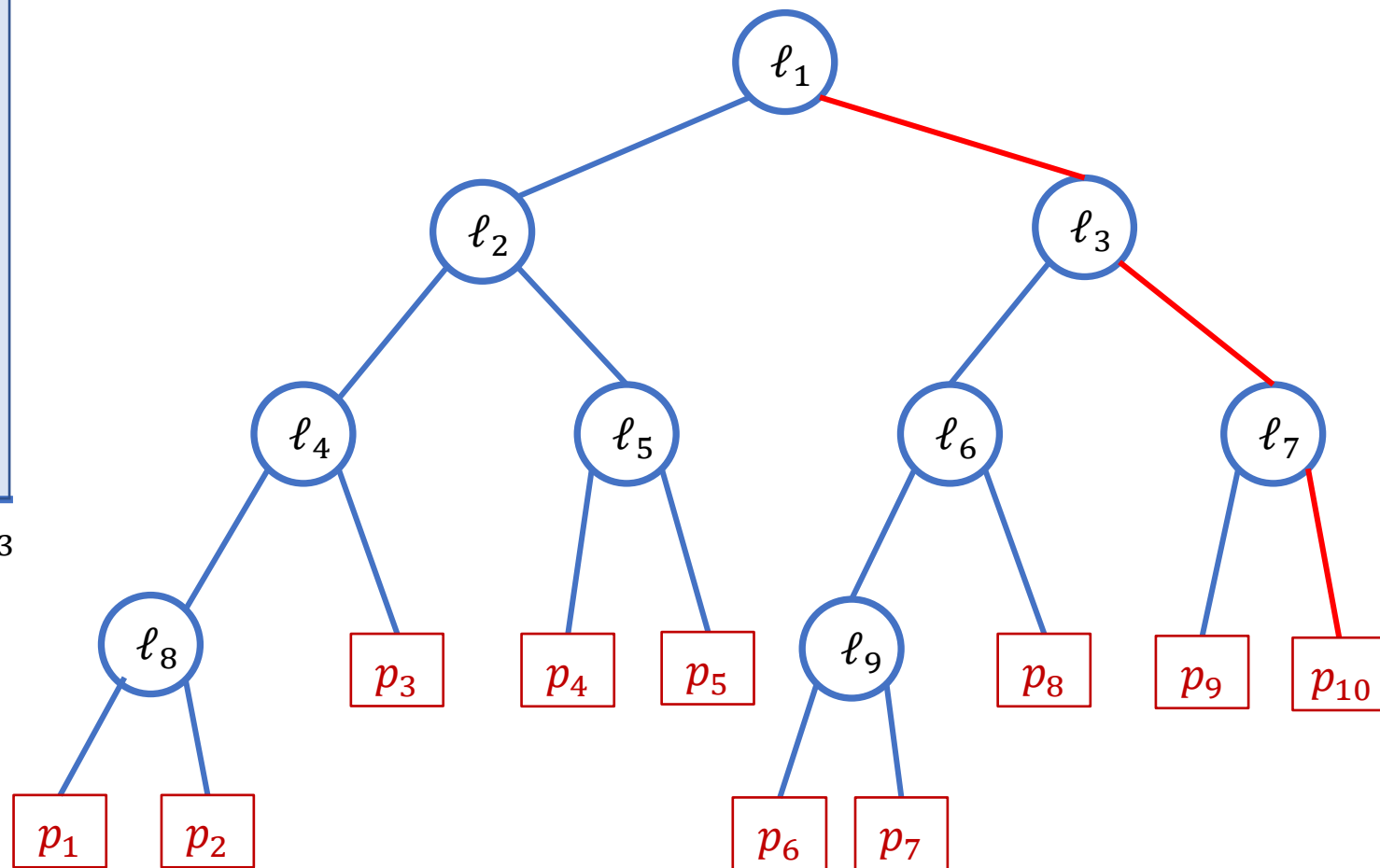
Current candidate:  $p_{10}$



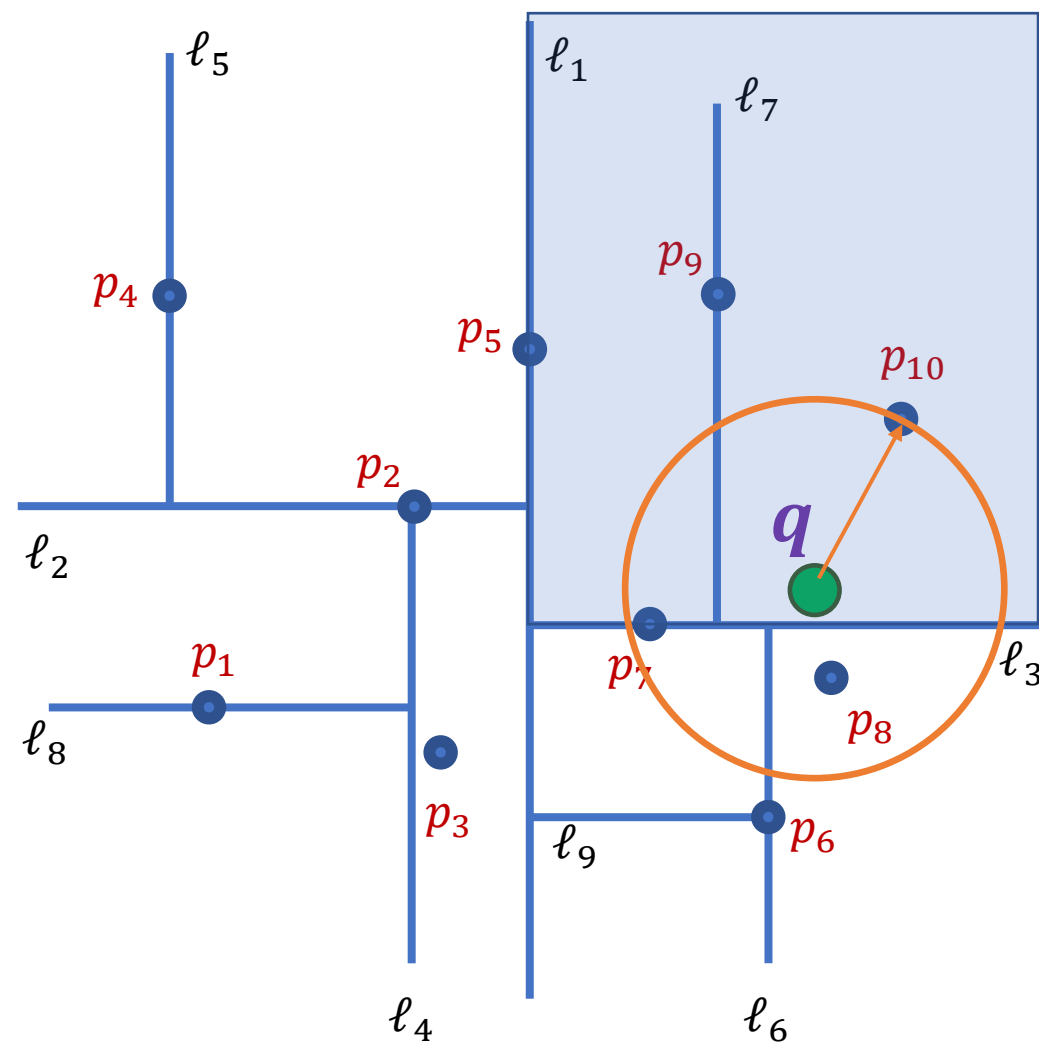
# Step 2



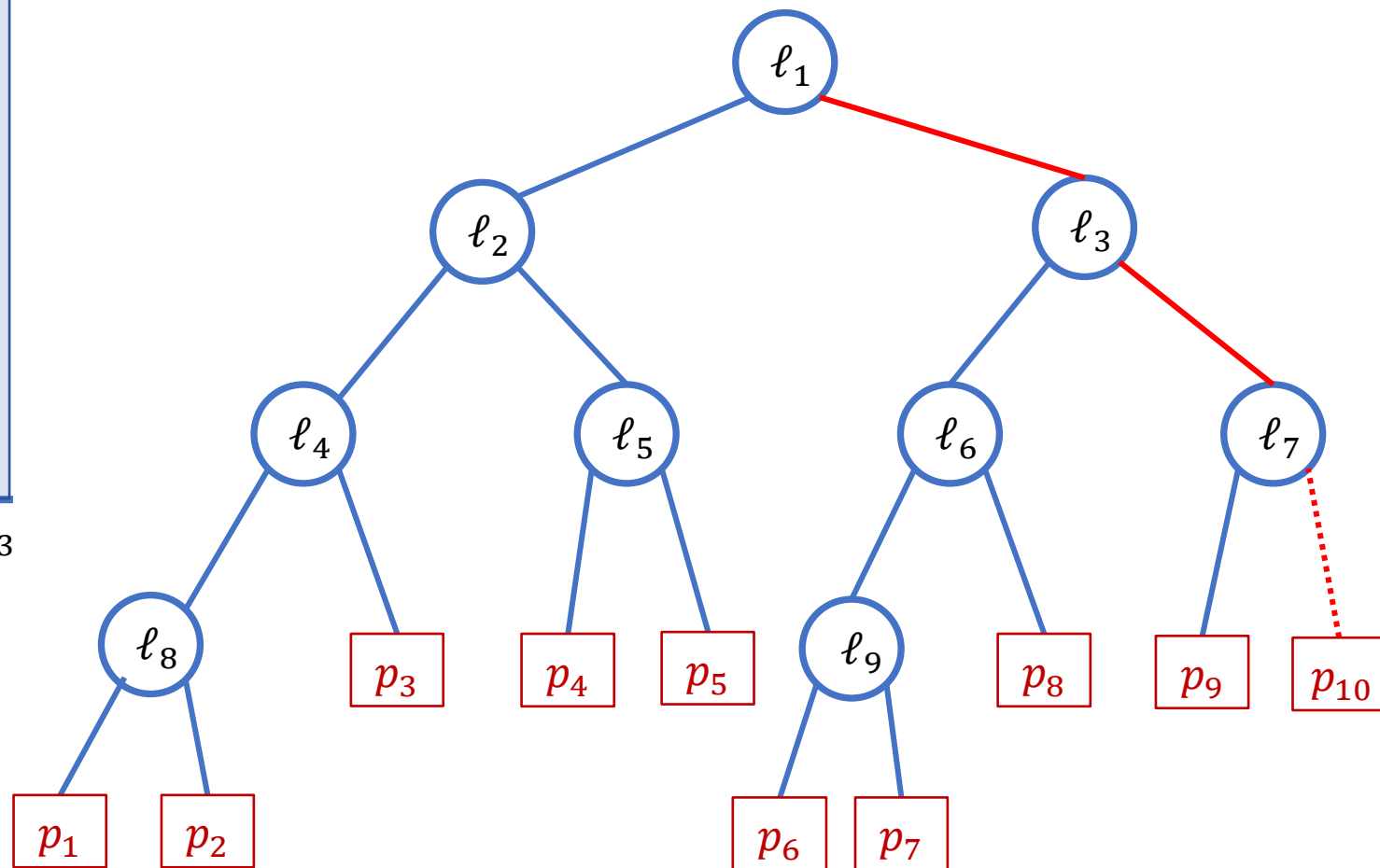
Current candidate:  $p_{10}$



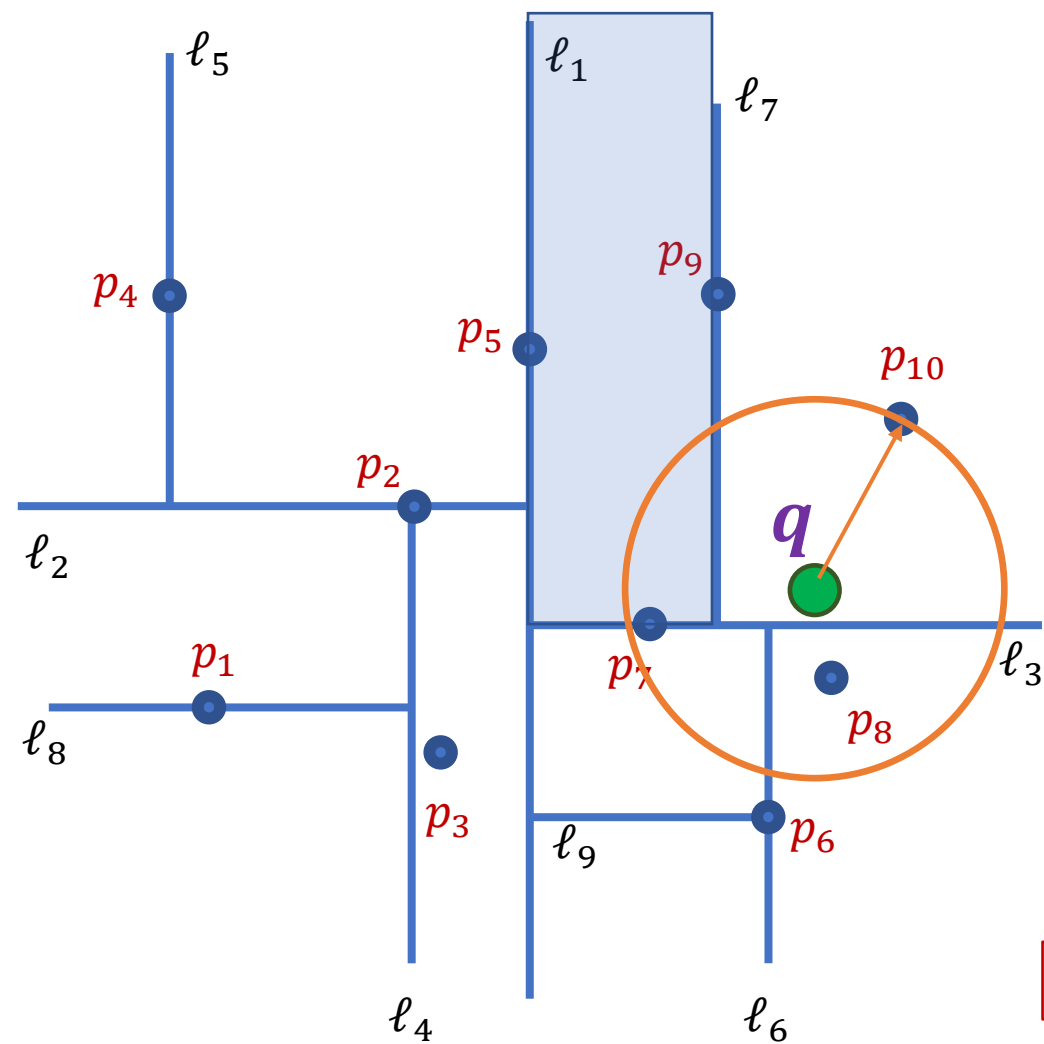
# Step 2



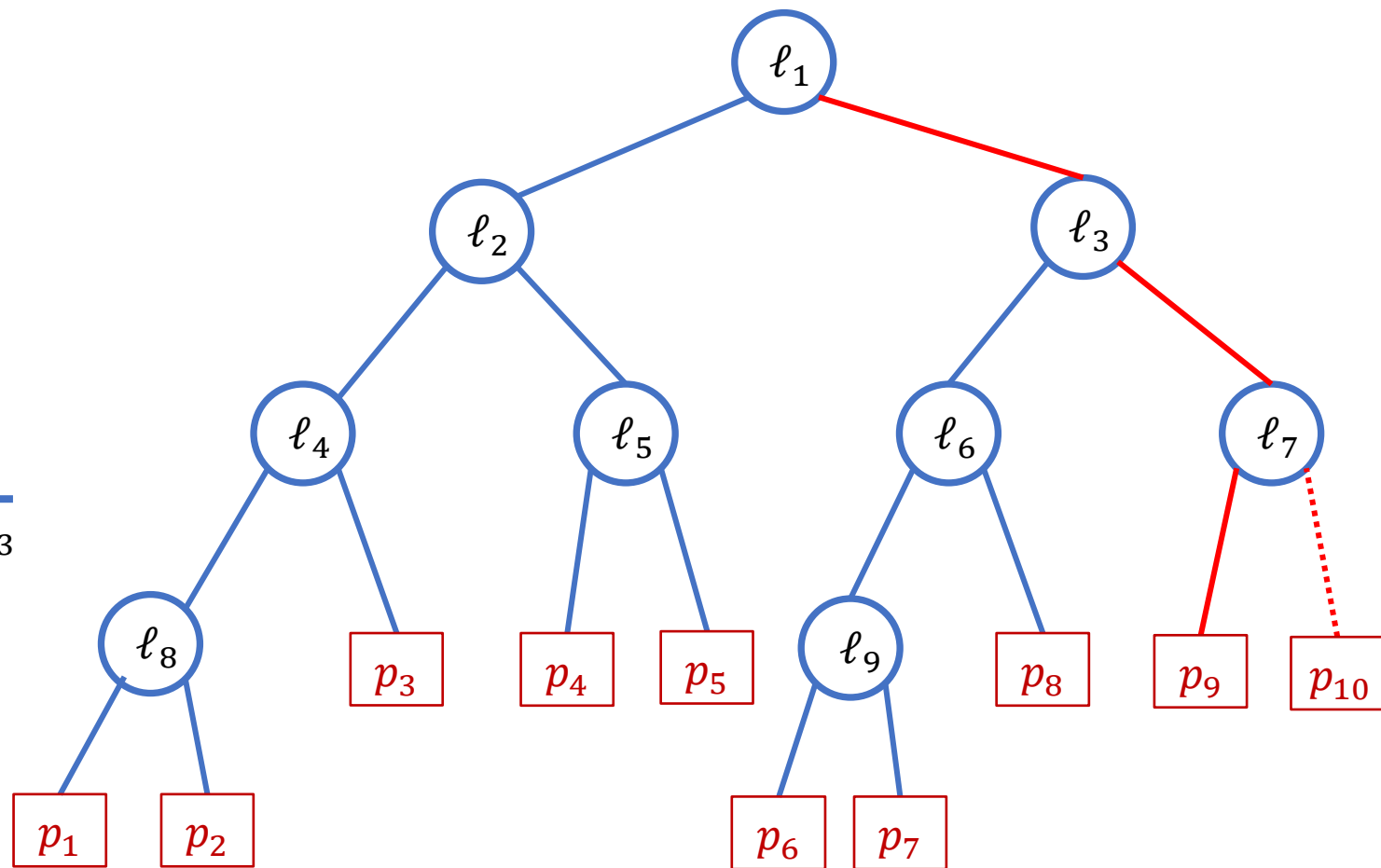
Current candidate:  $p_{10}$



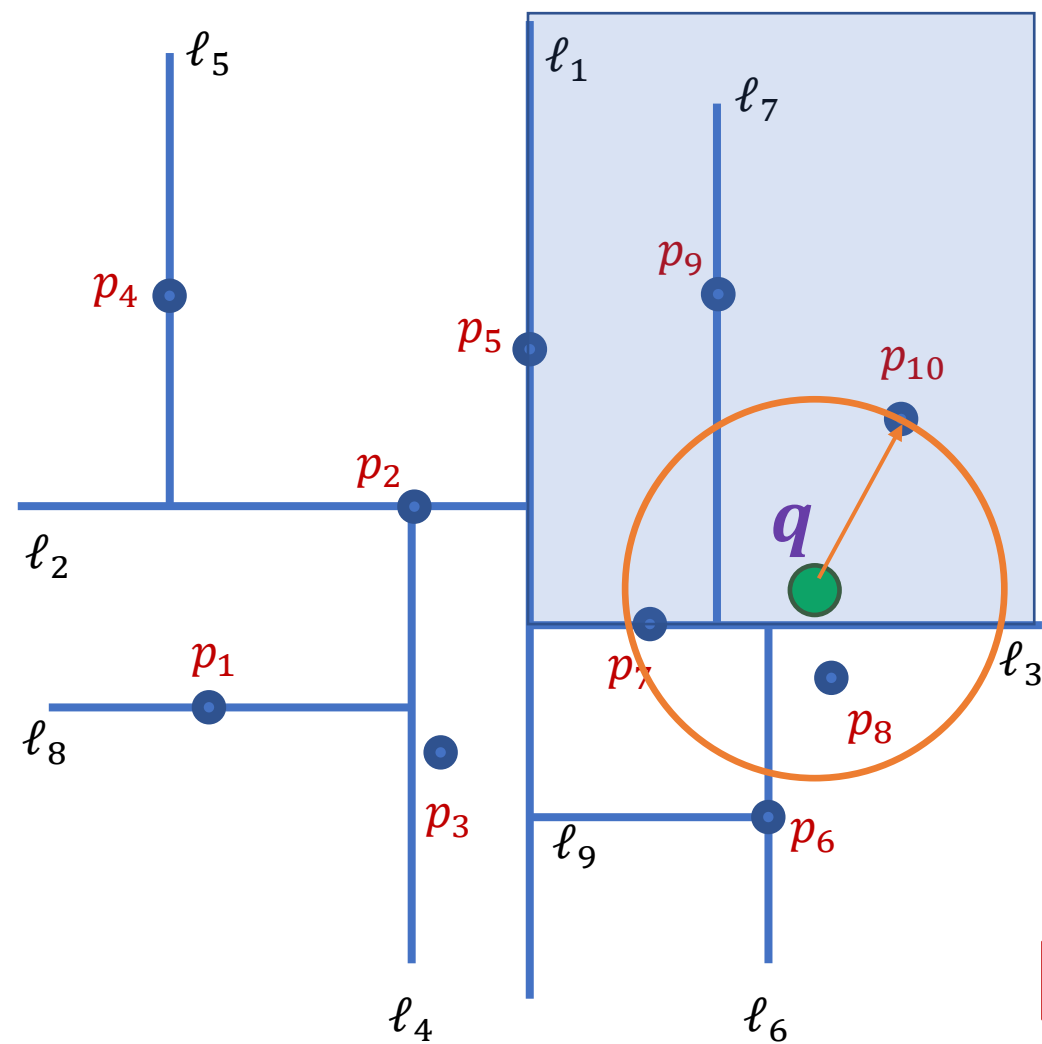
# Step 2



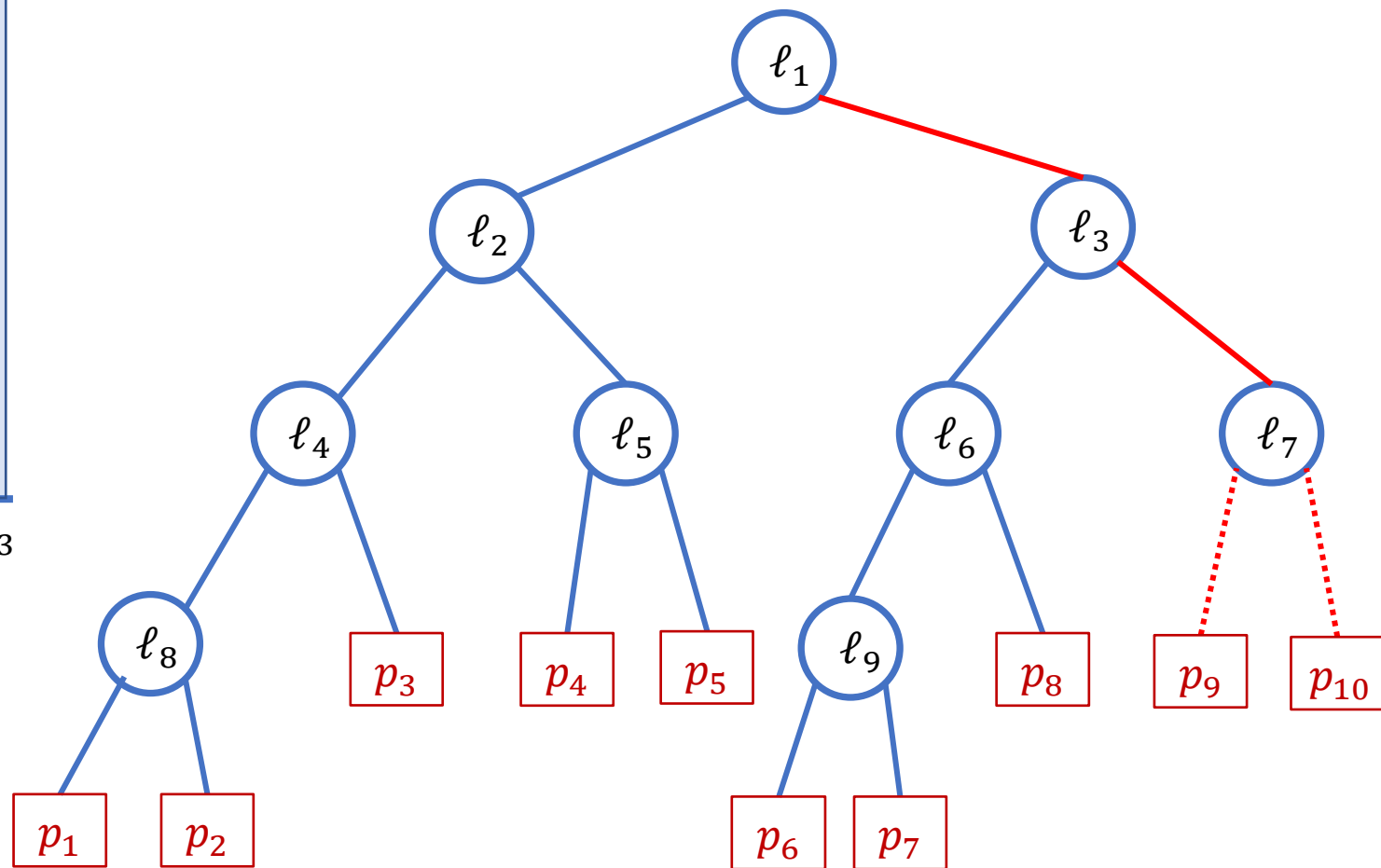
Current candidate:  $p_{10}$



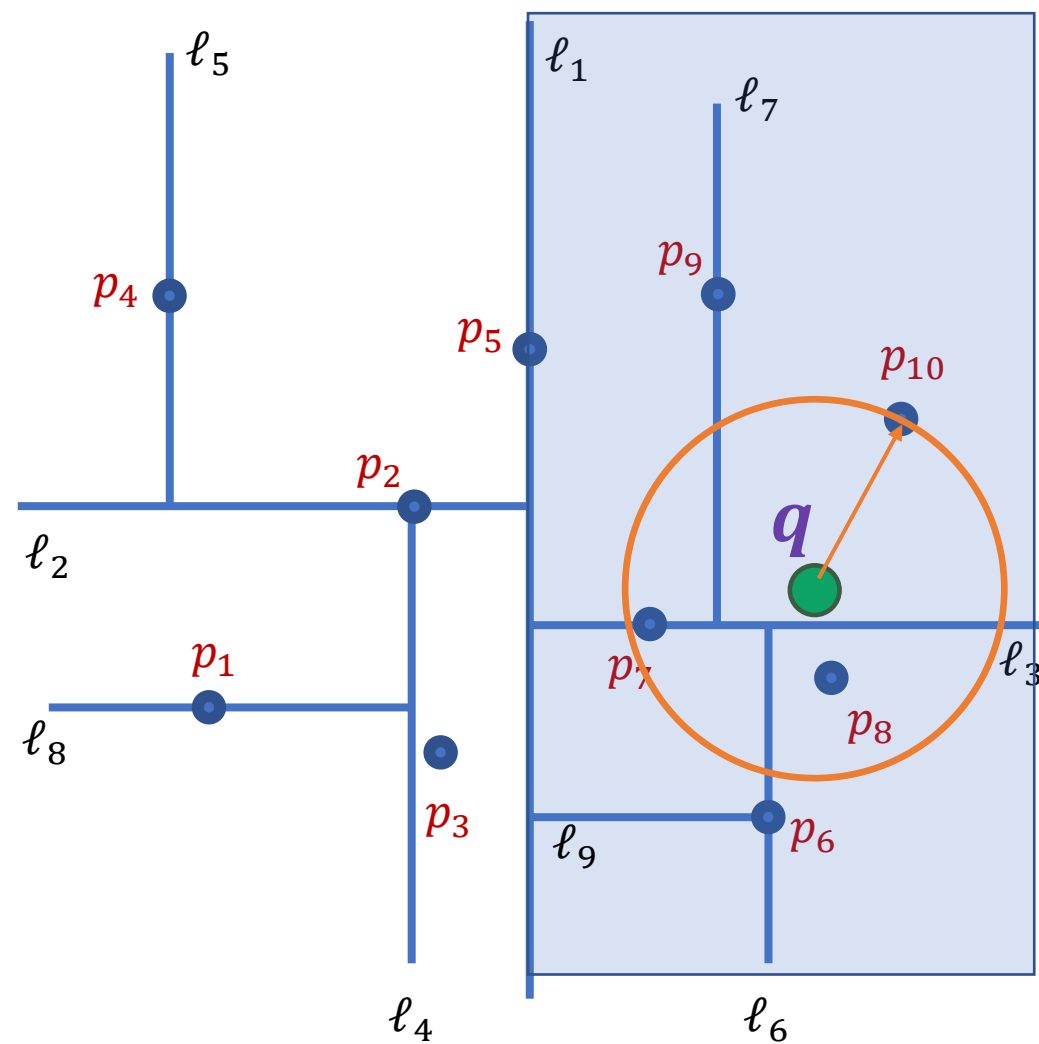
# Step 2



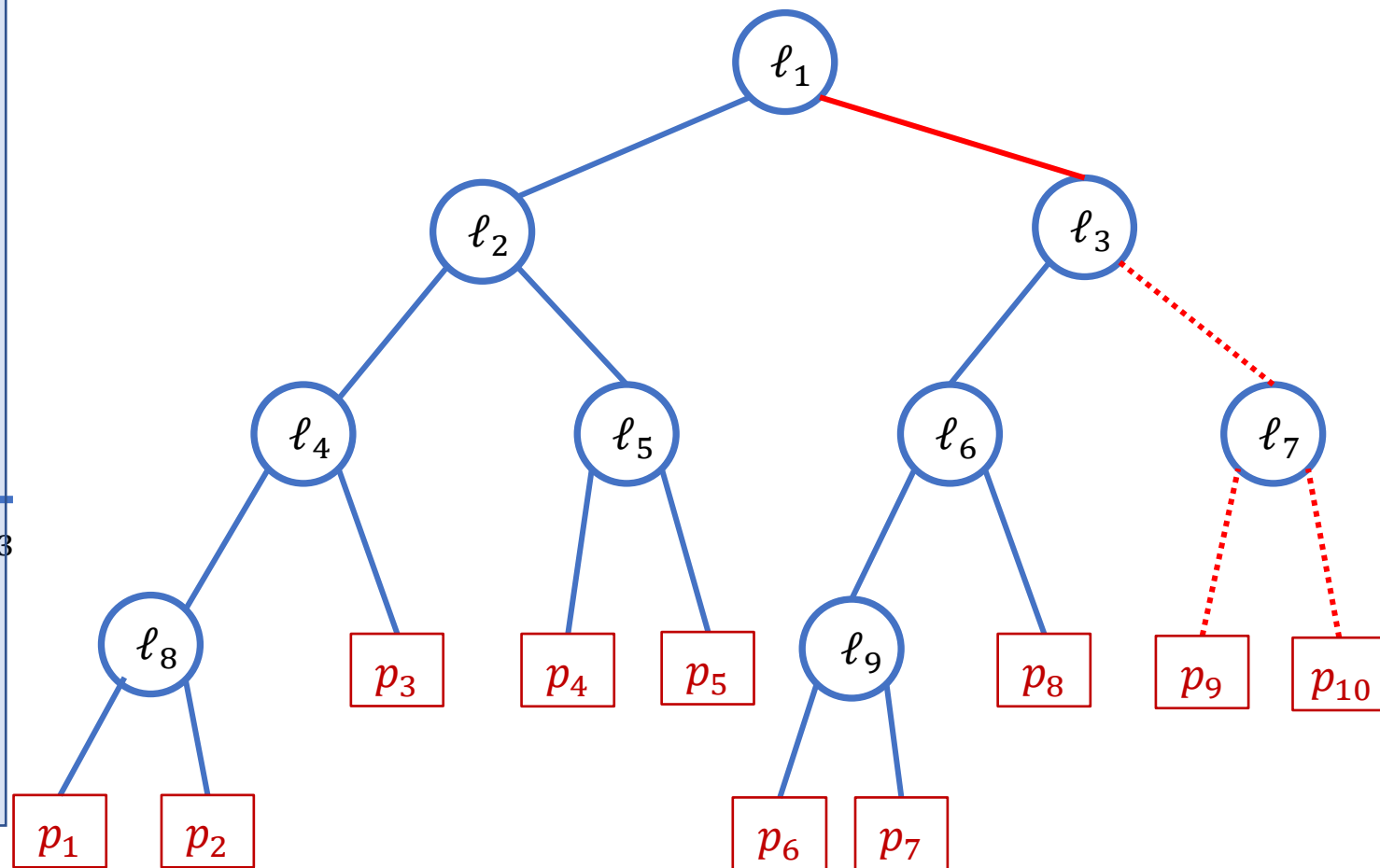
Current candidate:  $p_{10}$



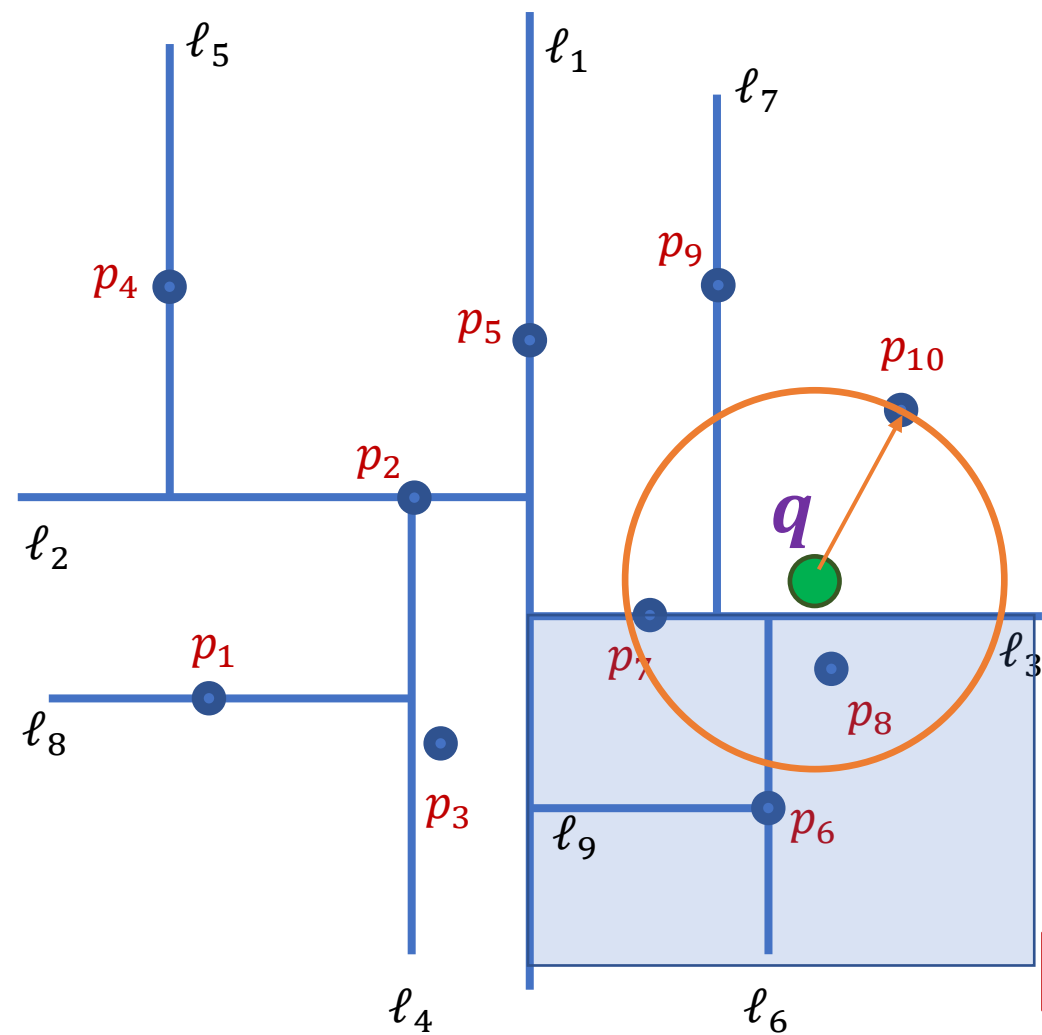
# Step 2



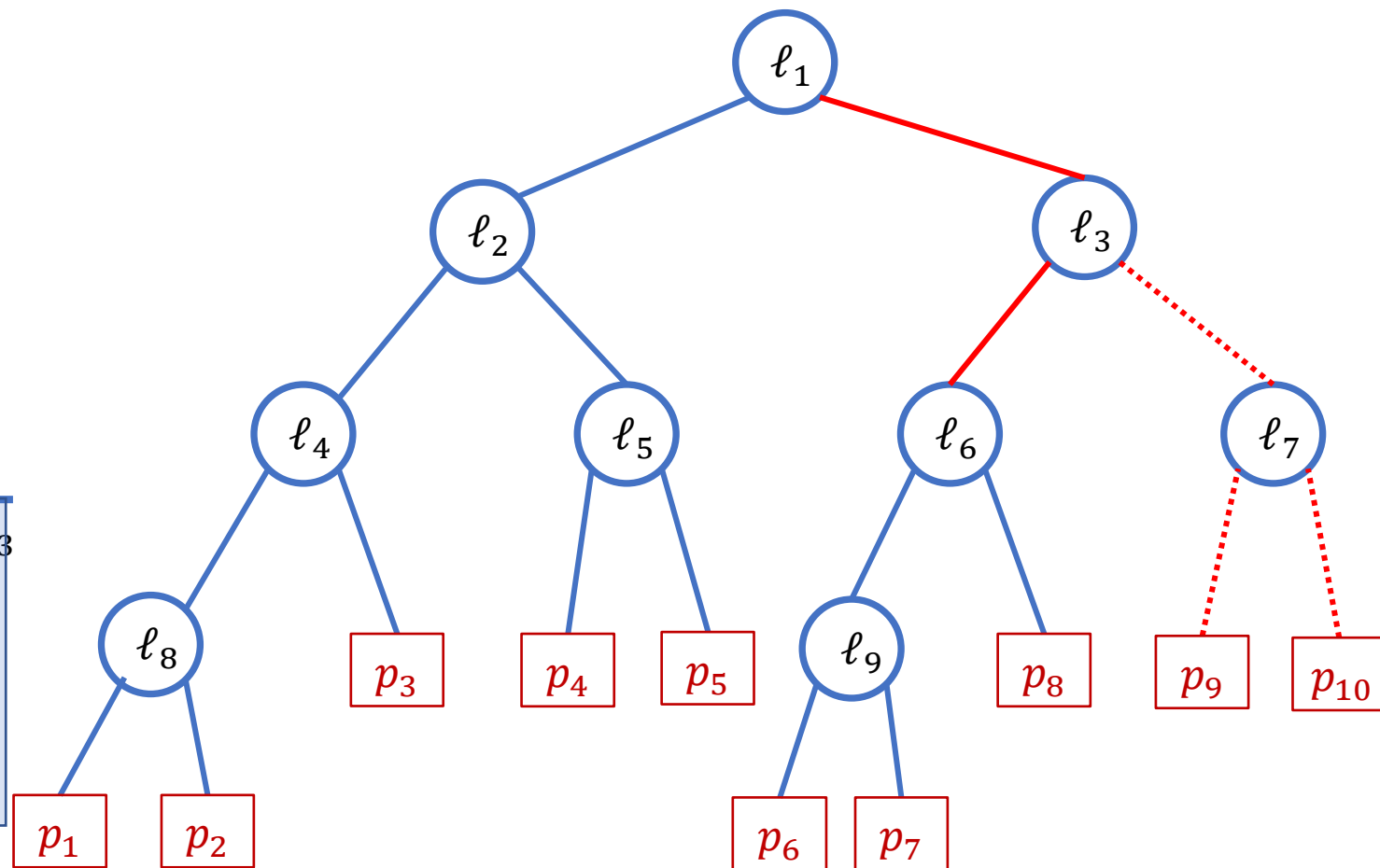
Current candidate:  $p_{10}$



# Step 2

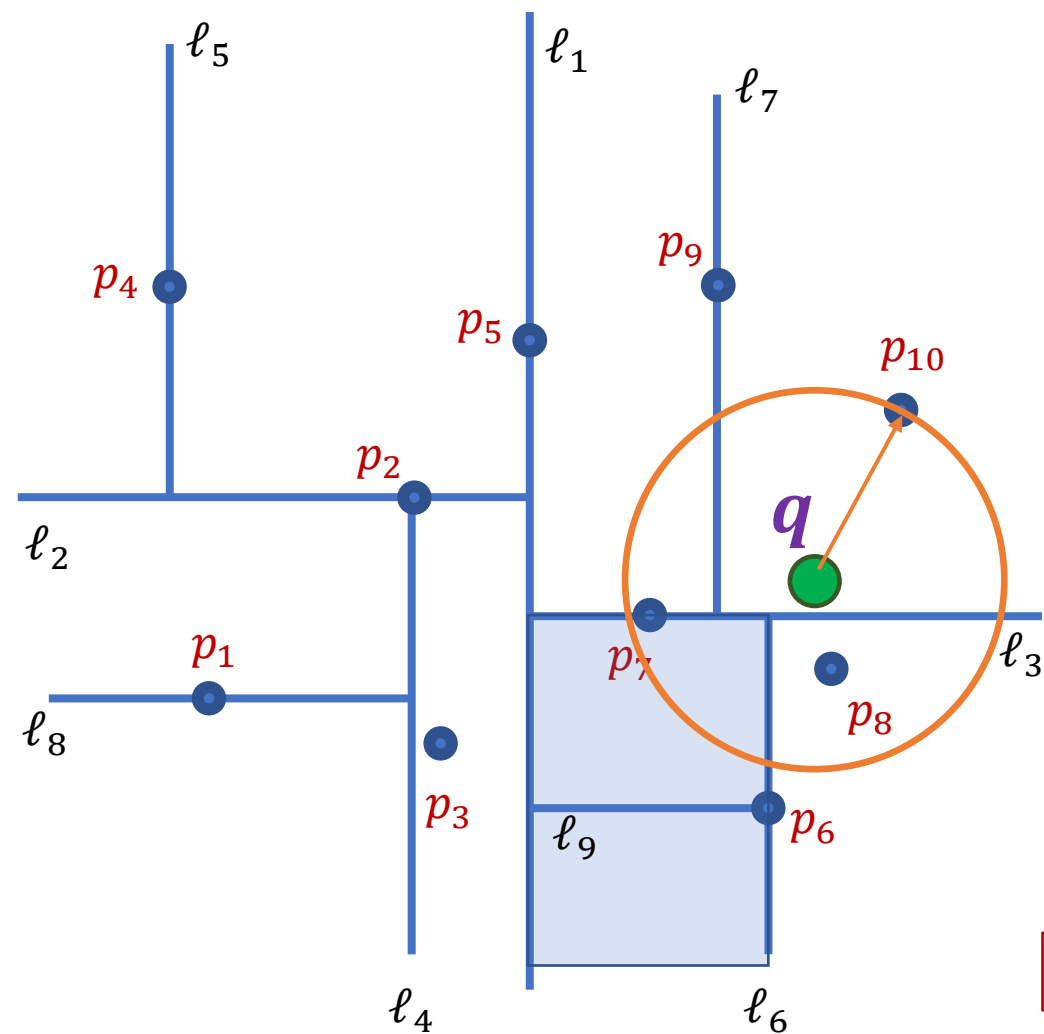


Current candidate:  $p_{10}$

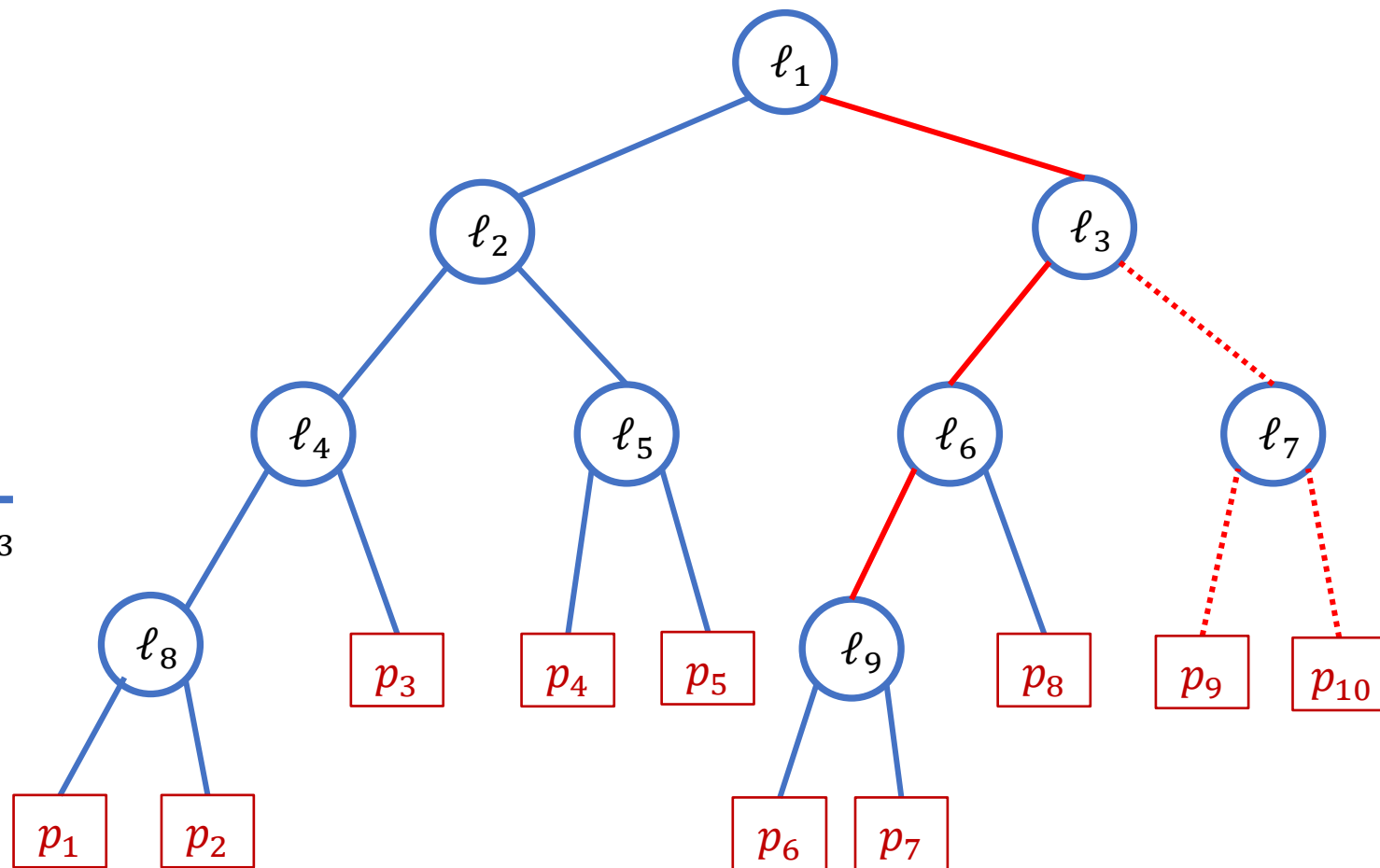




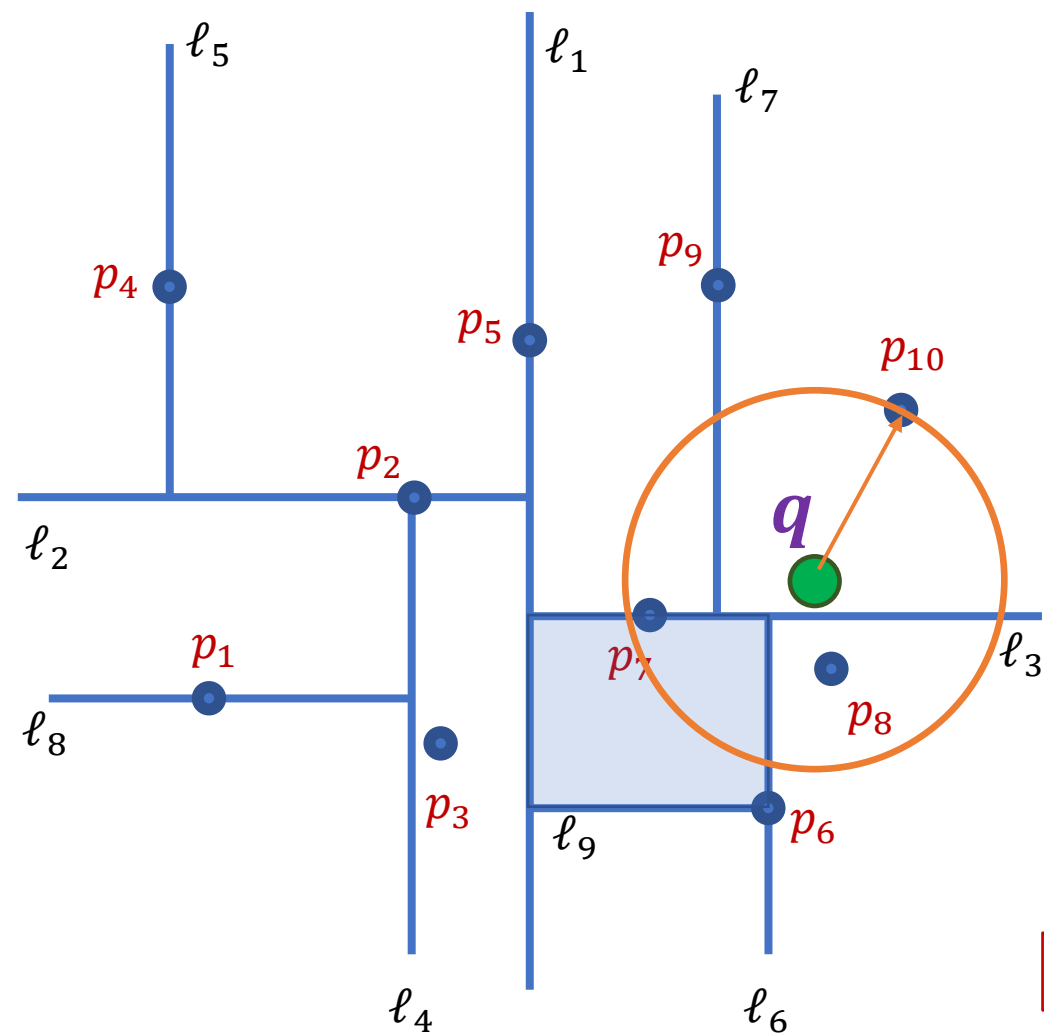
# Step 2



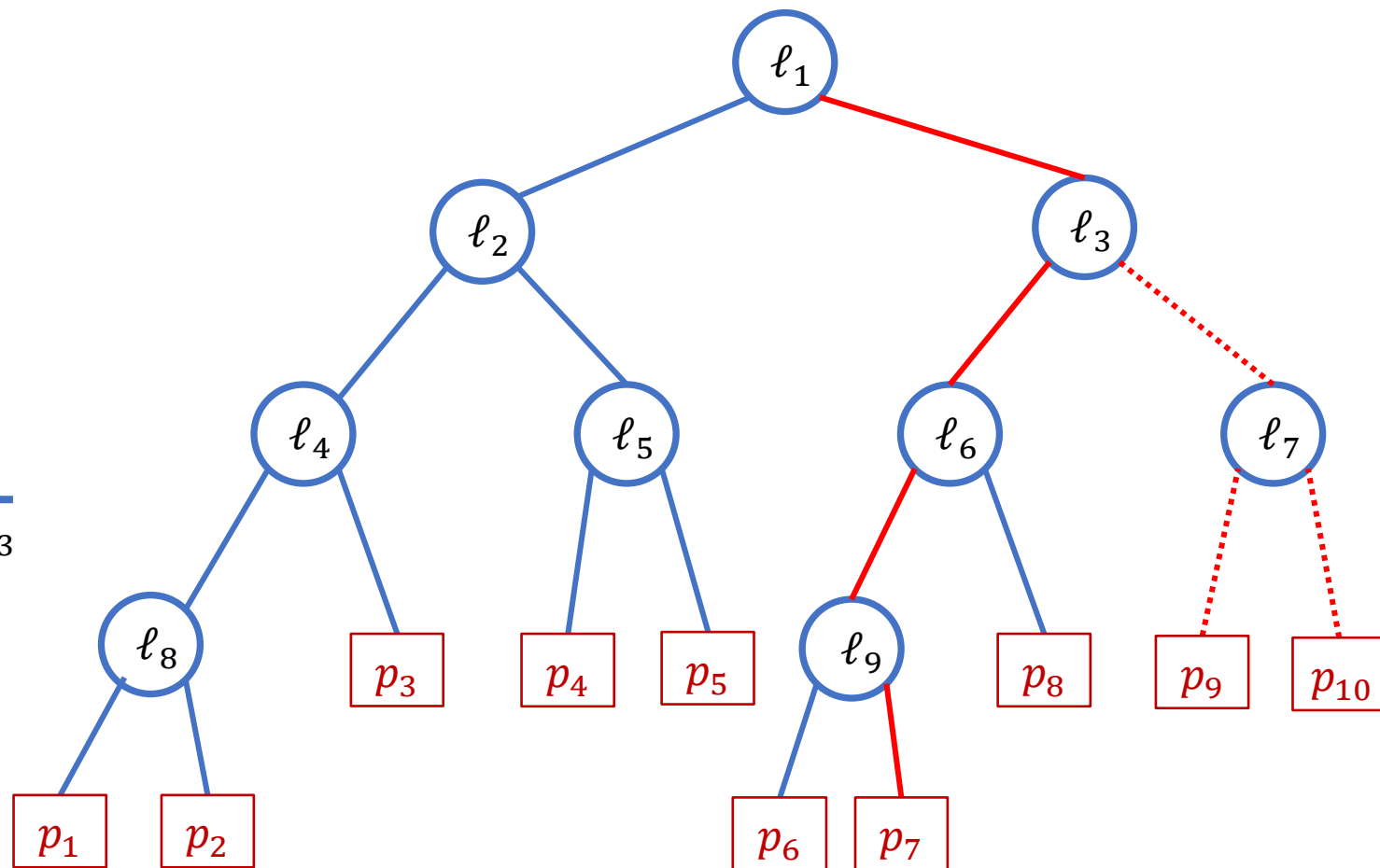
Current candidate:  $p_{10}$



# Step 2

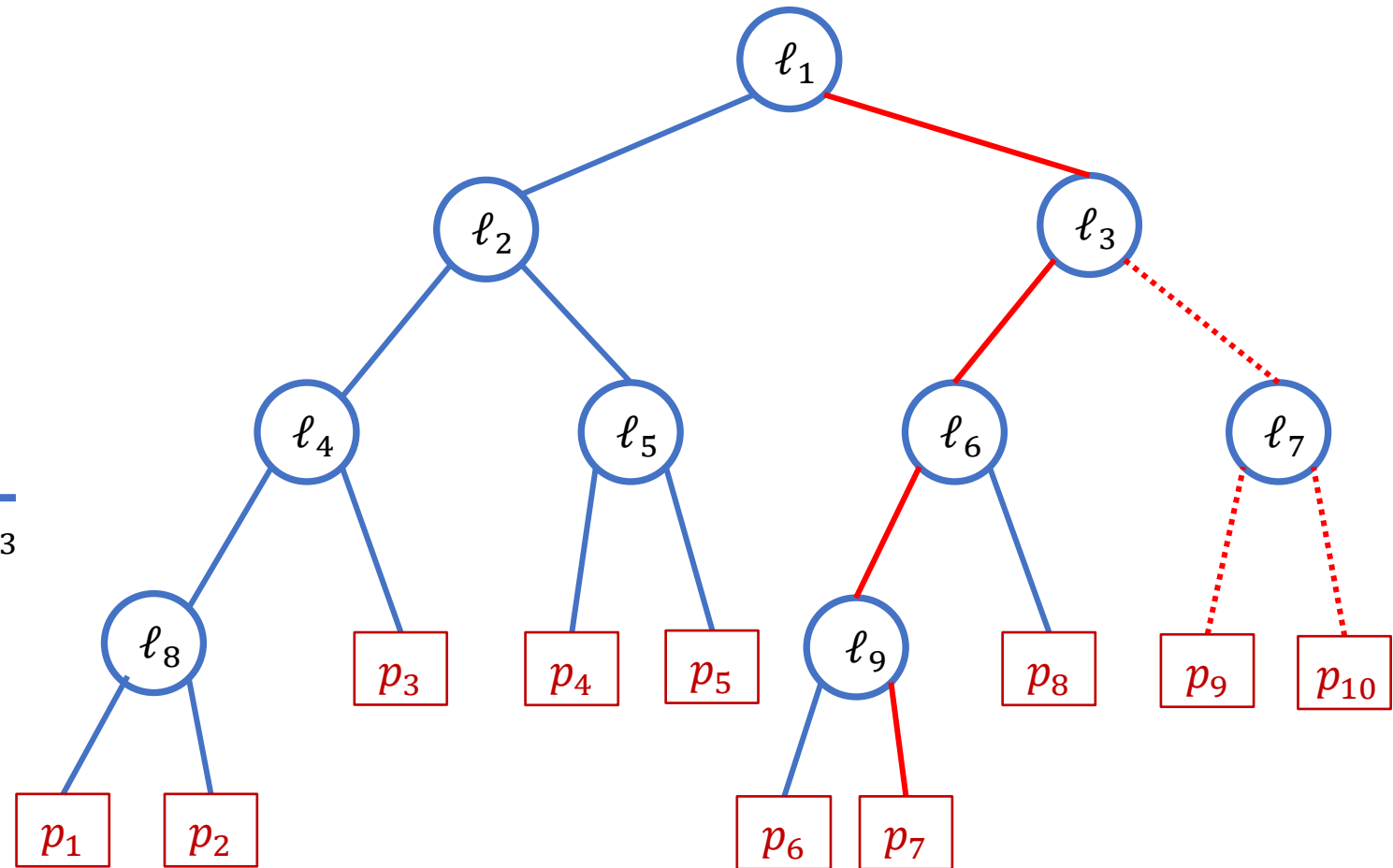
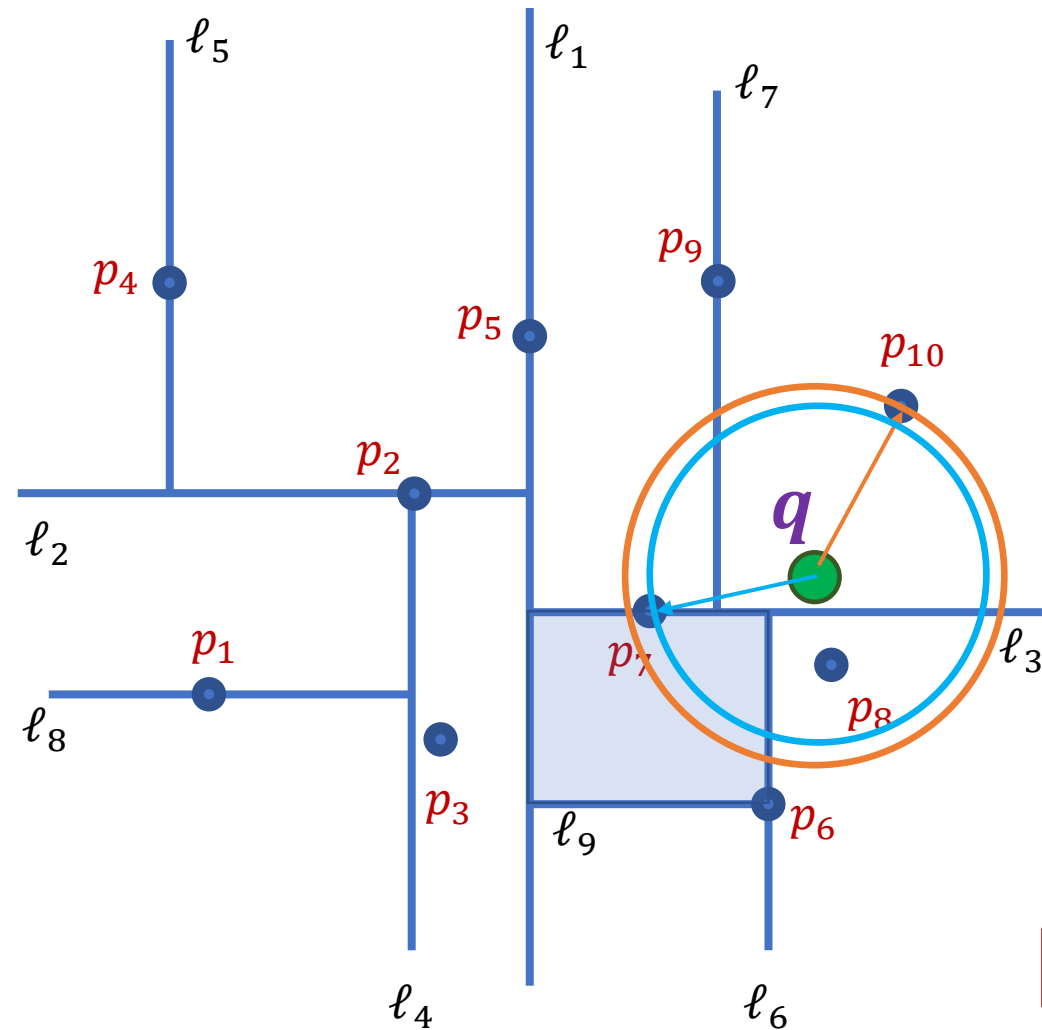


Current candidate:  $p_{10}$



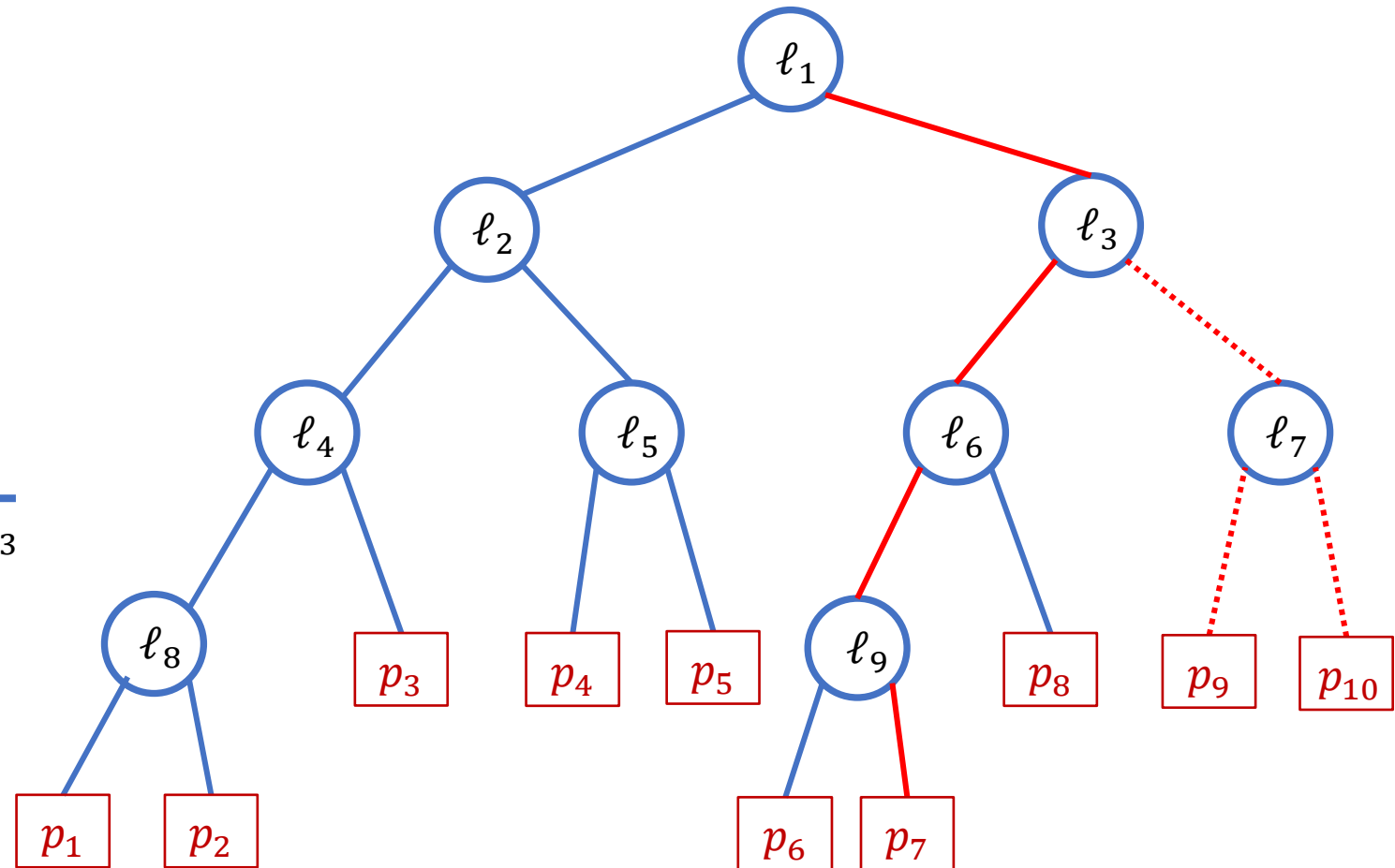
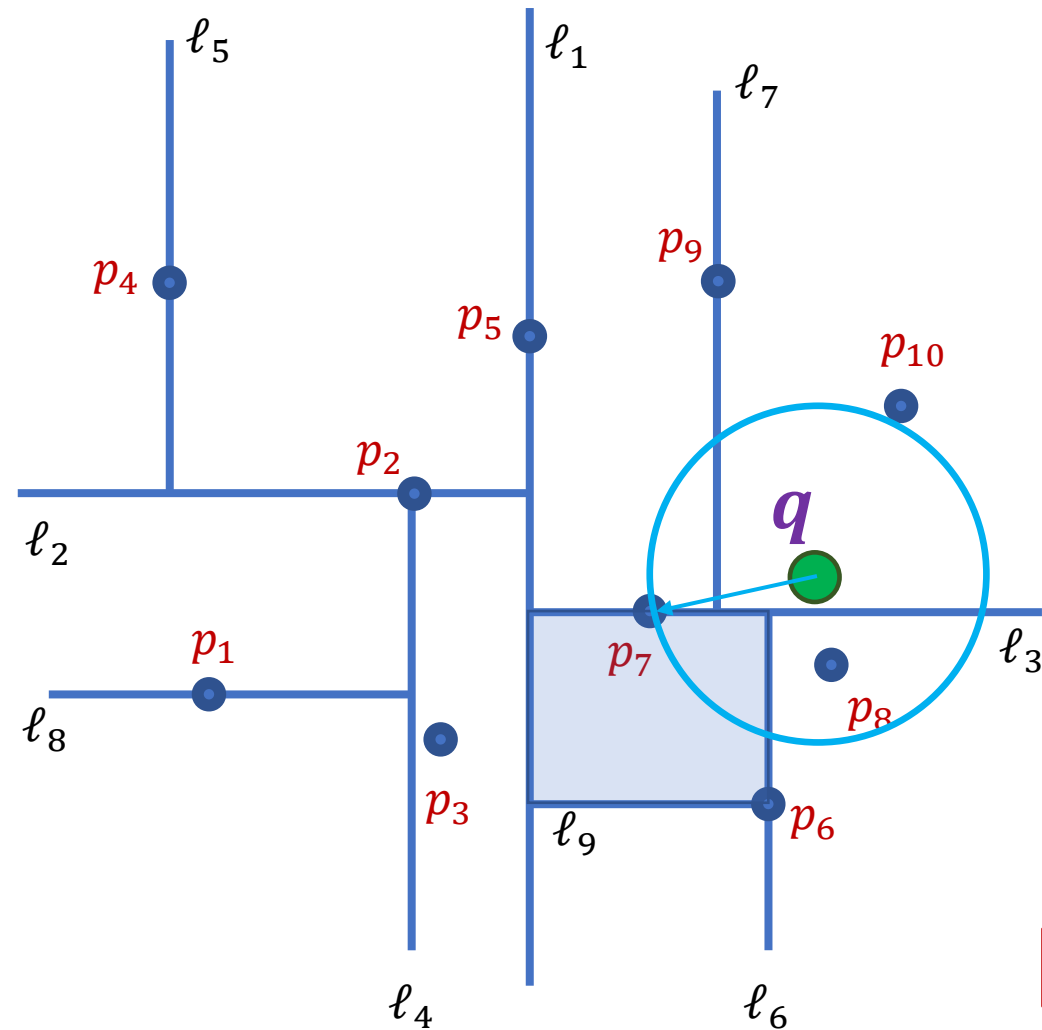
# Updating the candidate to $p_7$

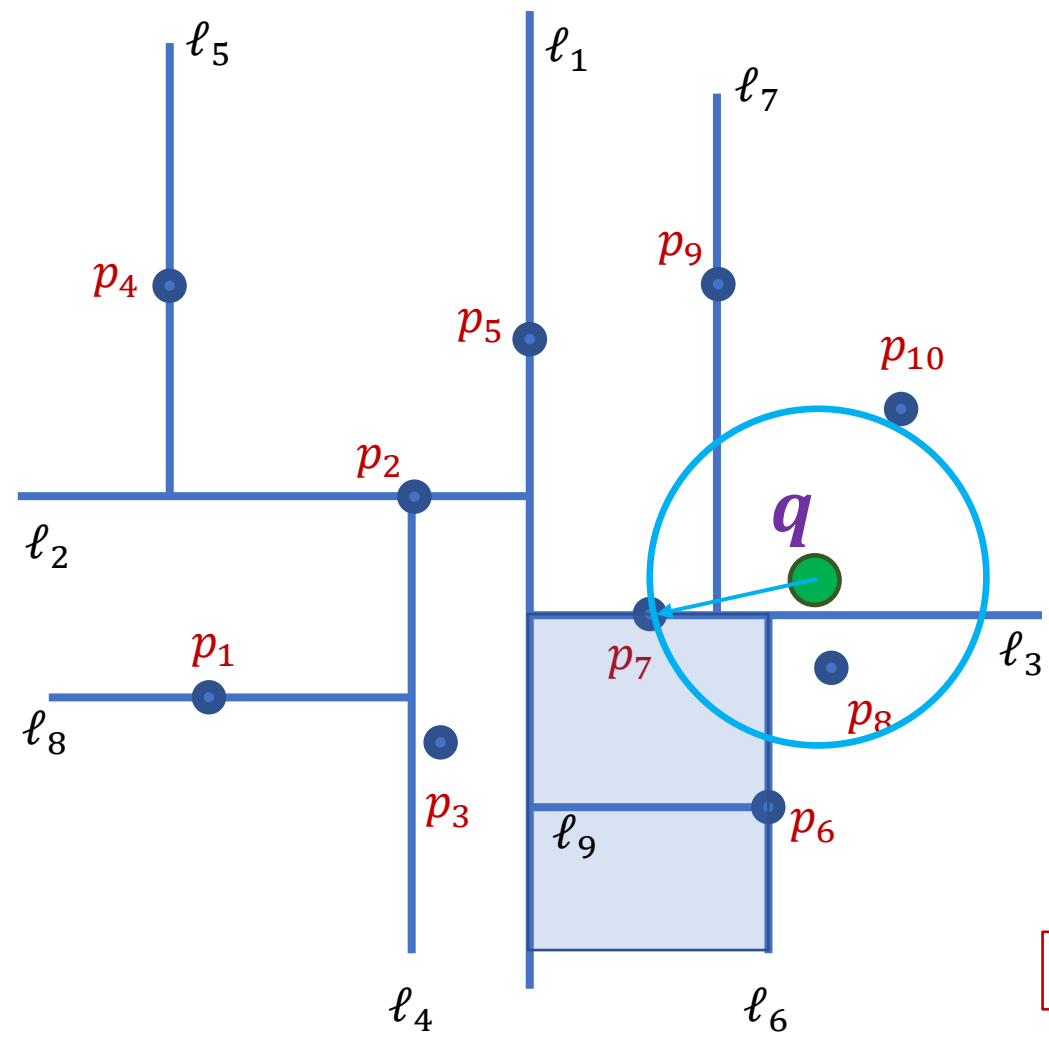
Current candidate:  $p_7$



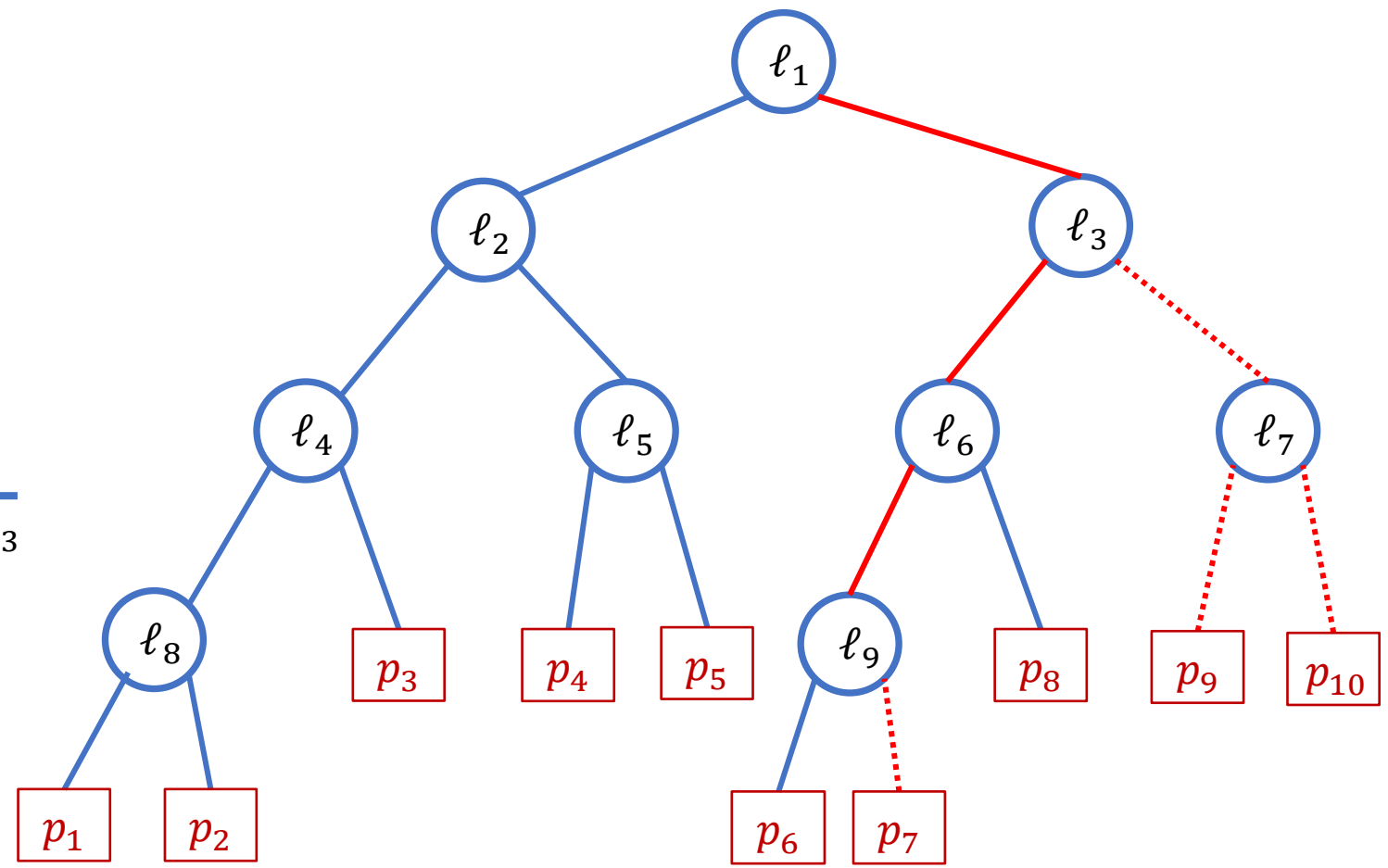
# Updating the candidate to $p_7$

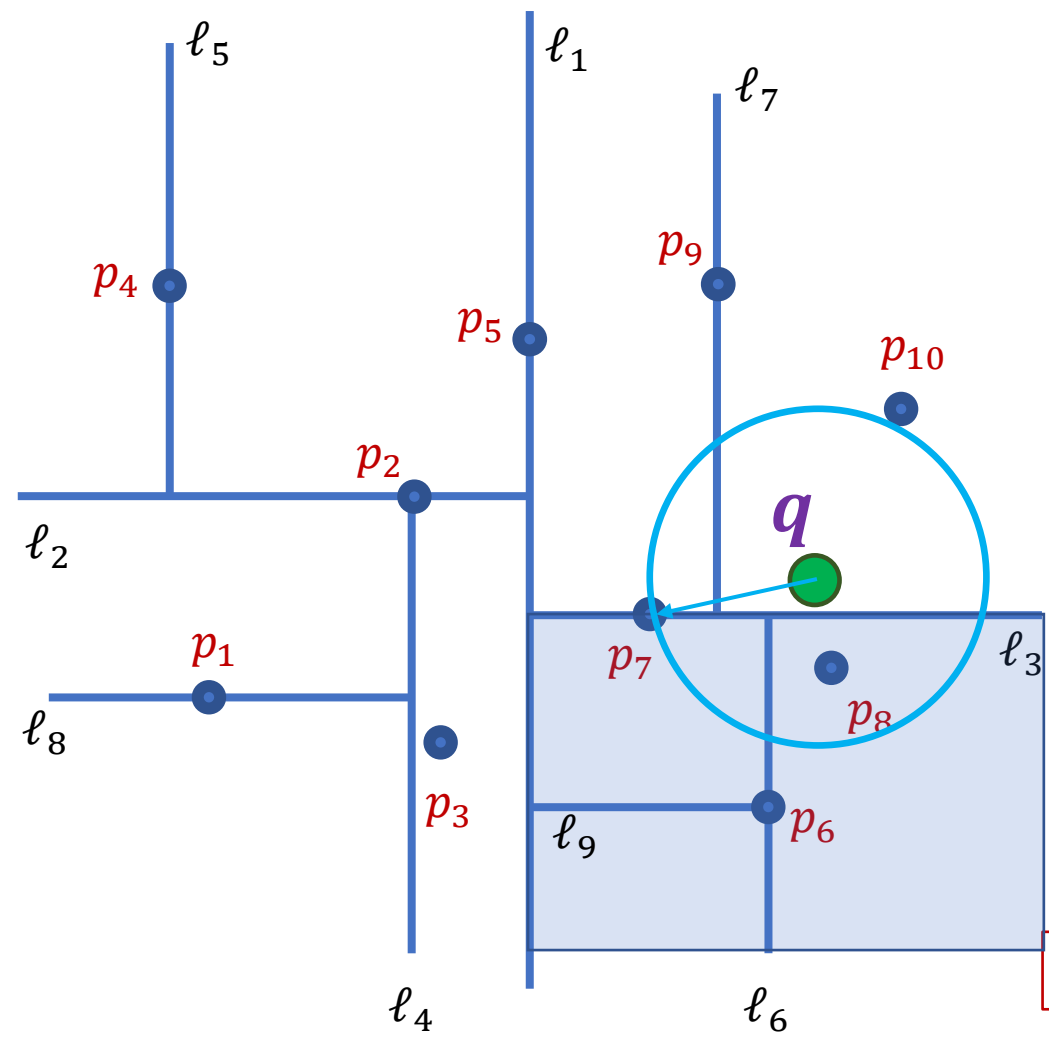
Current candidate:  $p_7$



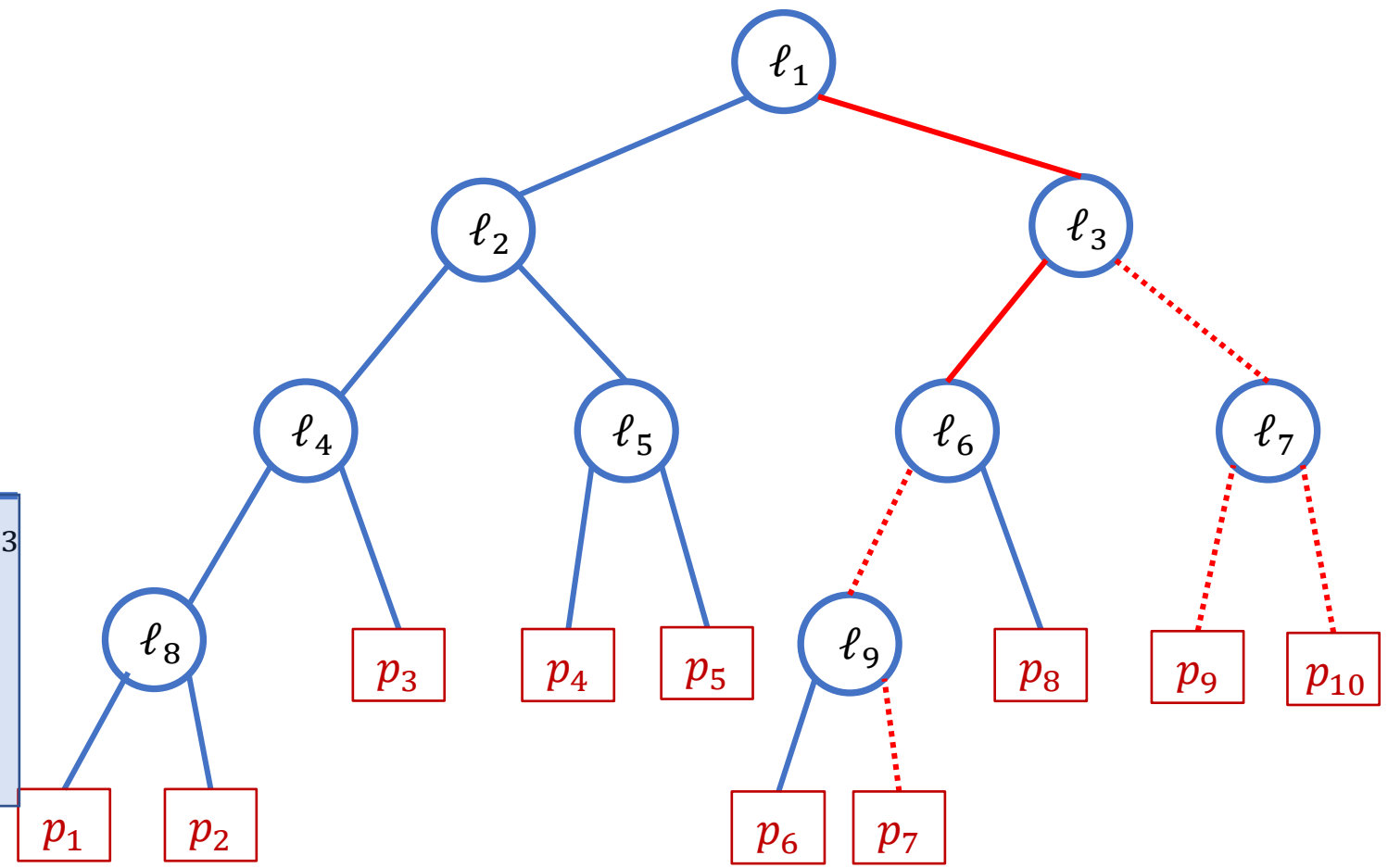


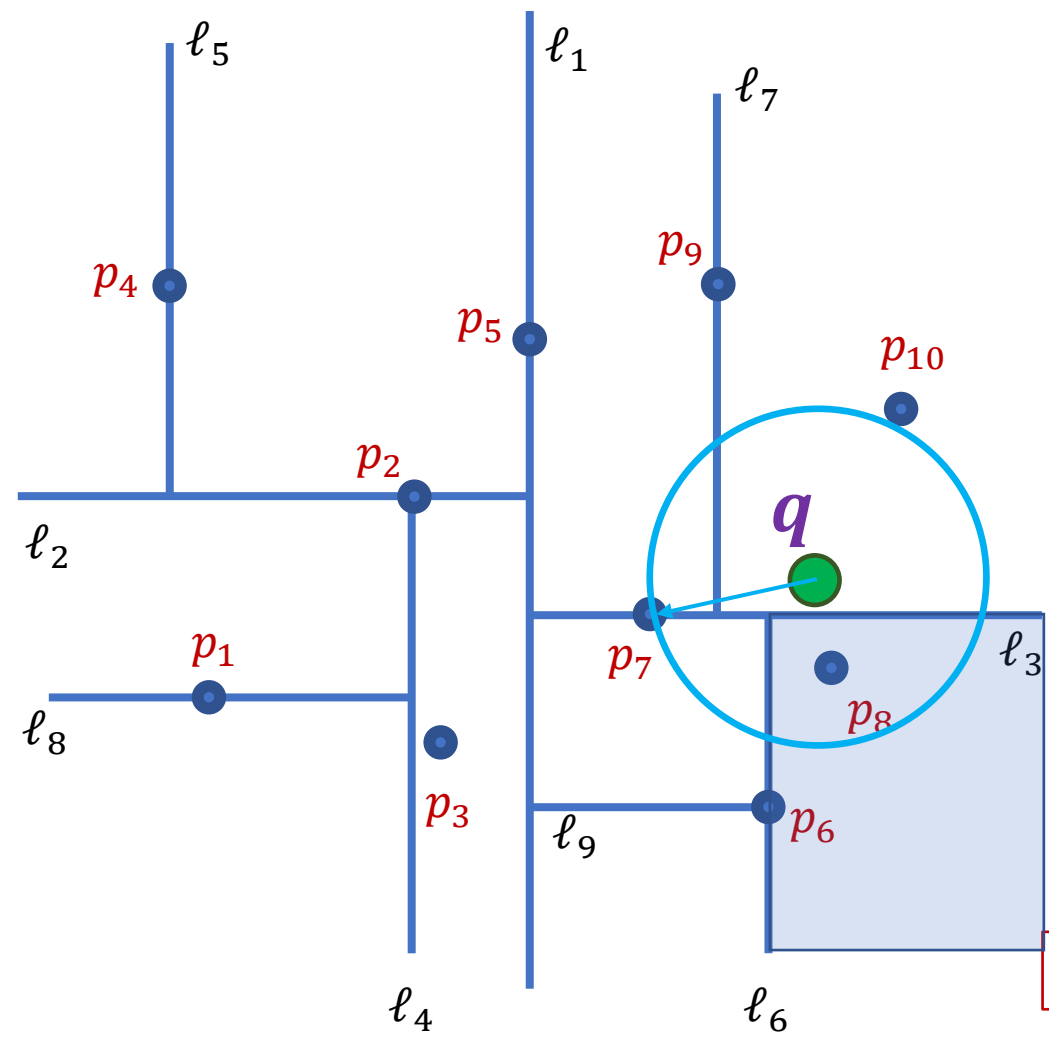
Current candidate:  $p_7$



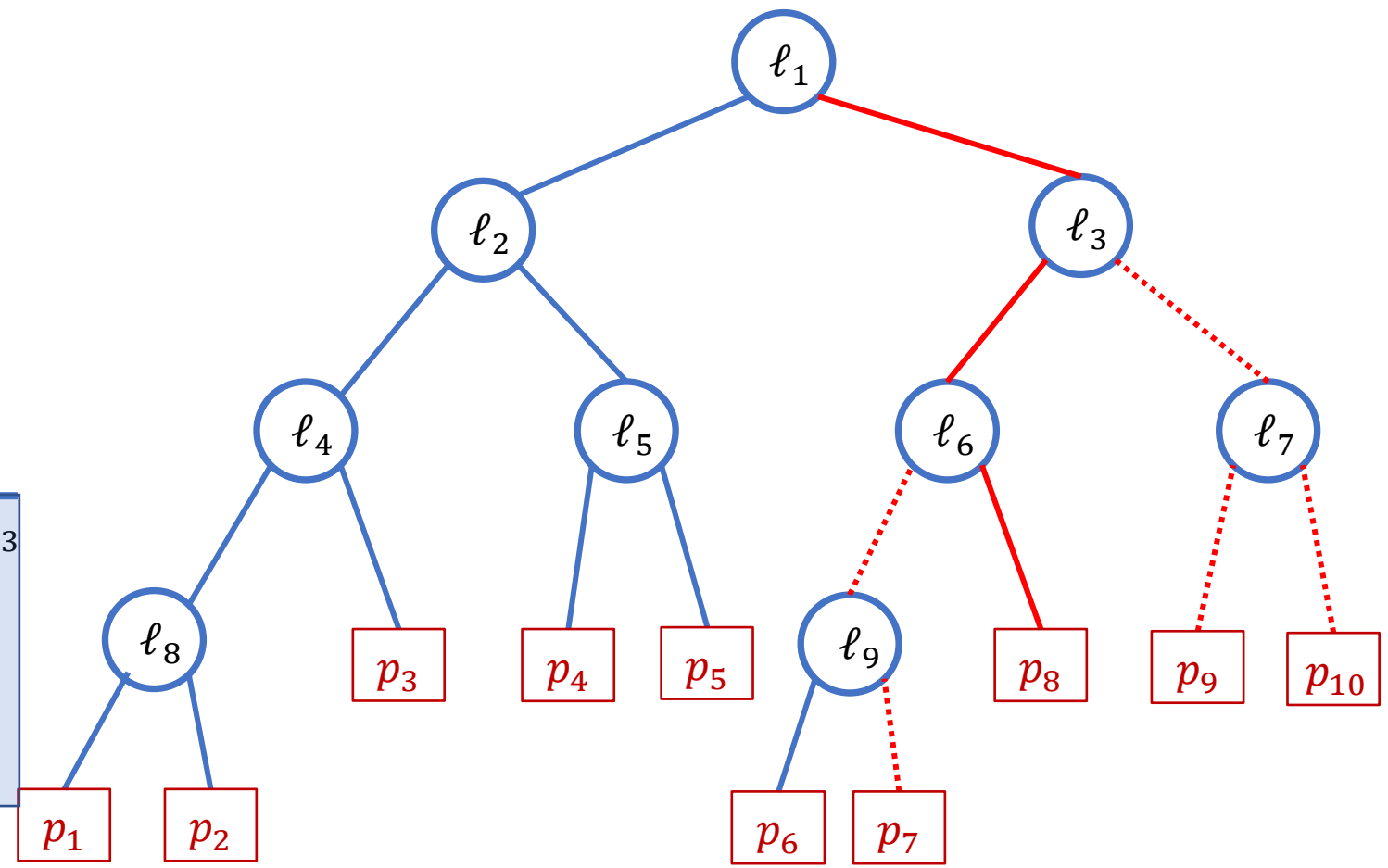


Current candidate:  $p_7$



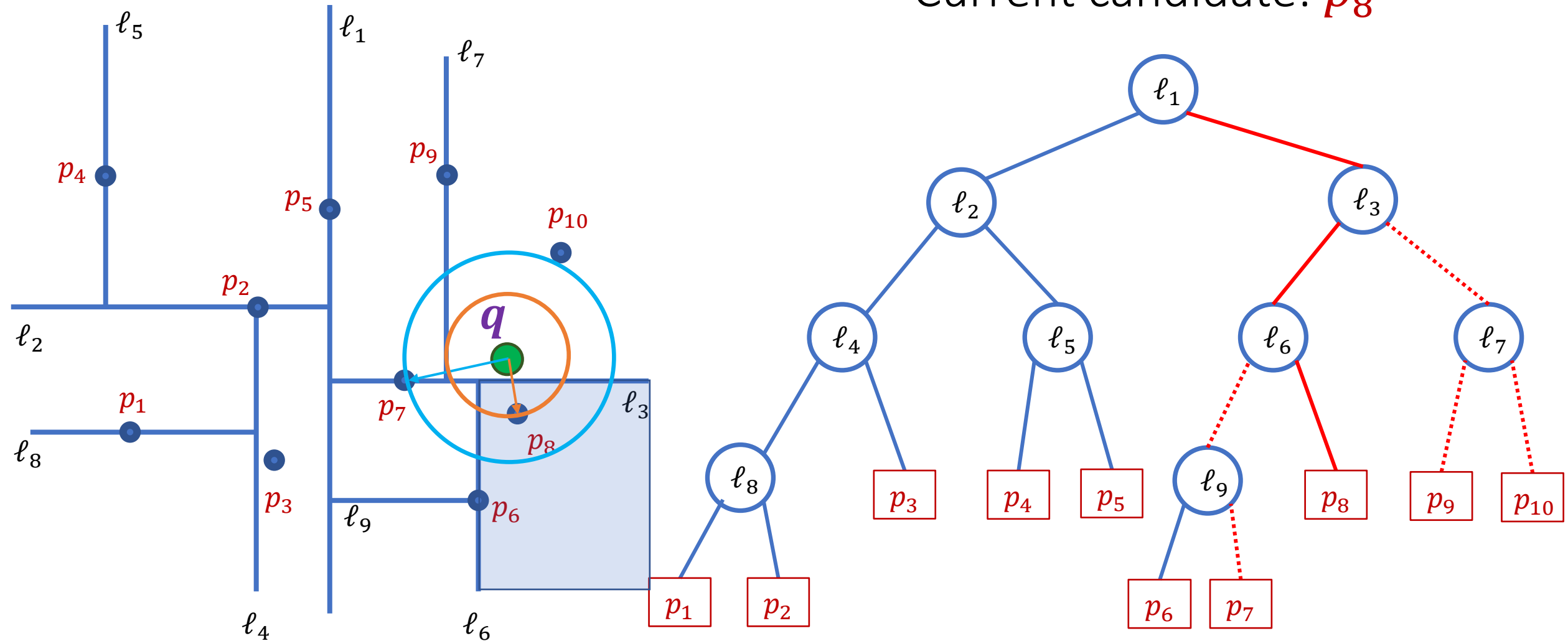


Current candidate:  $p_7$



# Updating the candidate to $p_8$

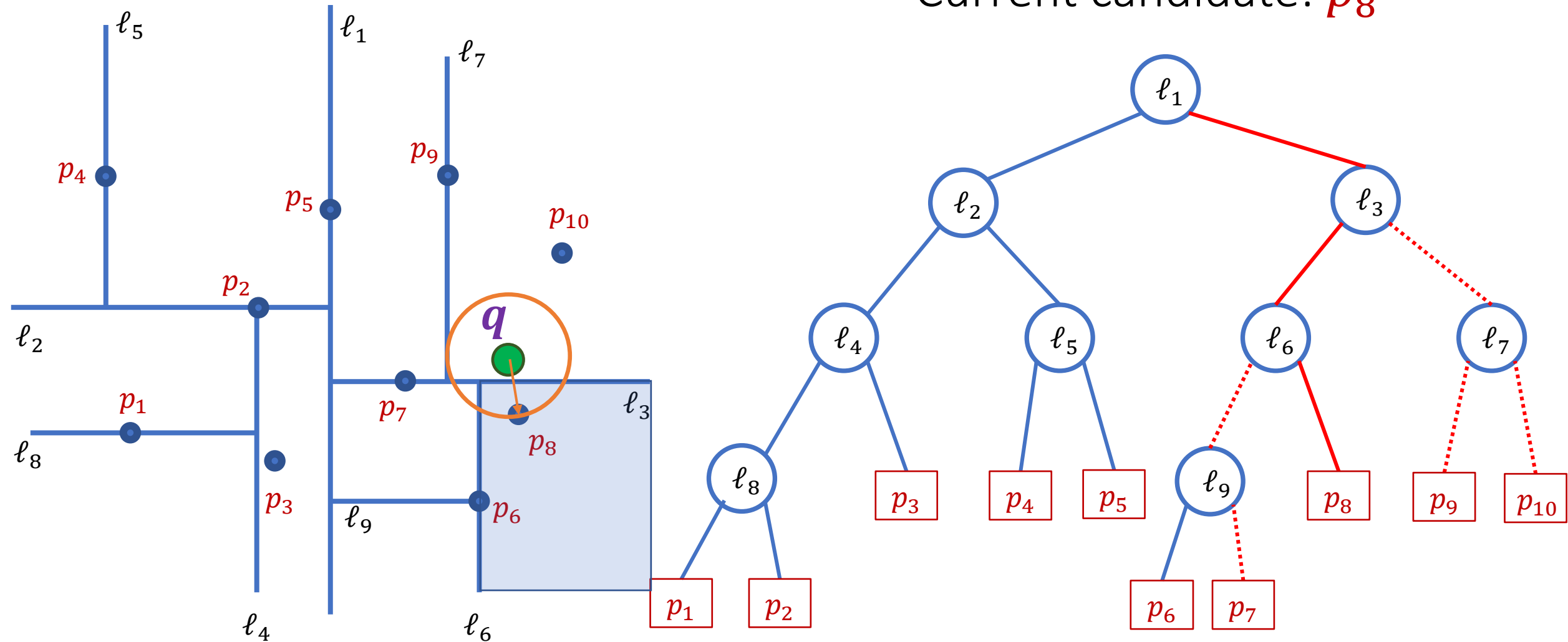
Current candidate:  $p_8$



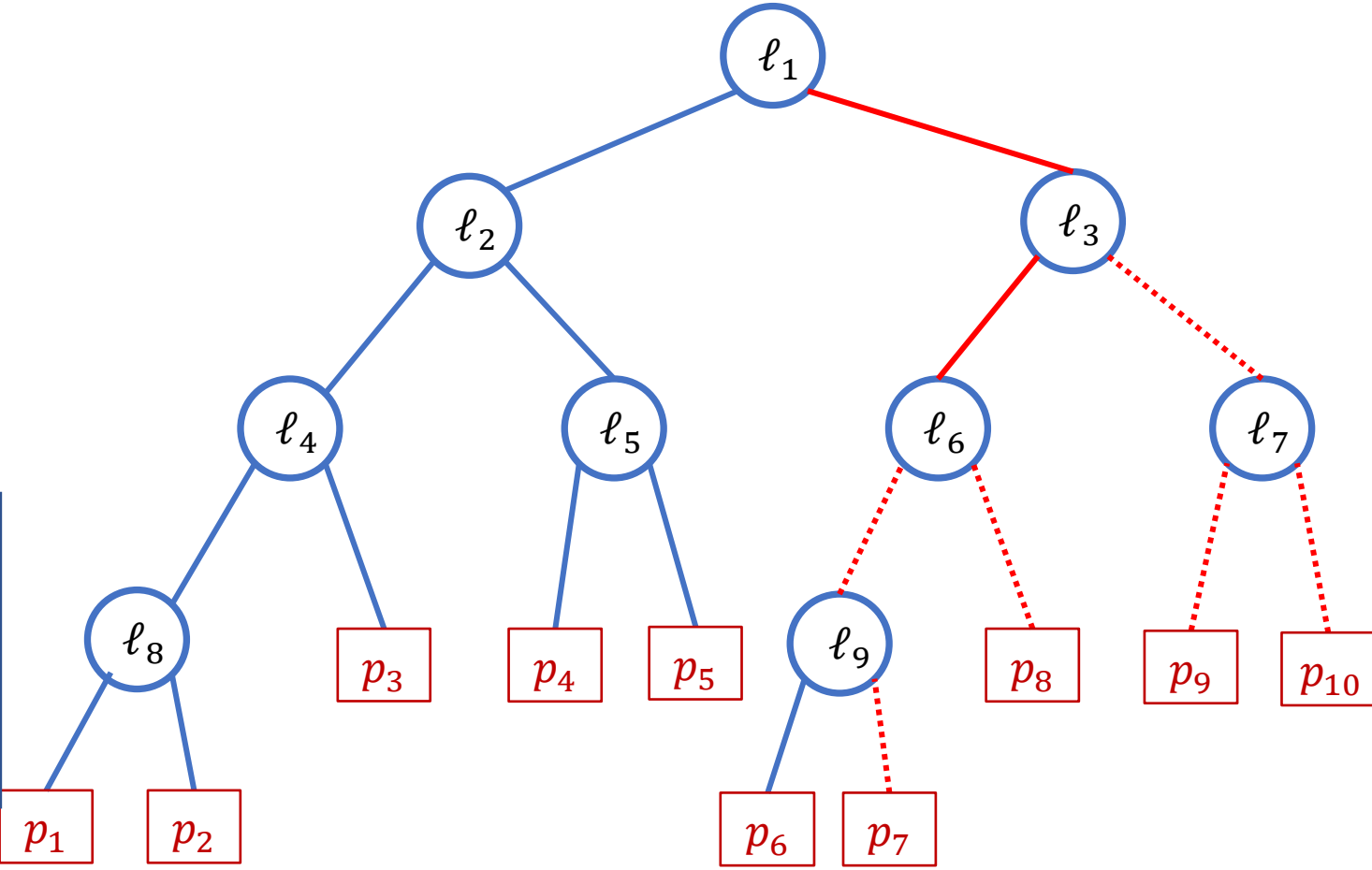
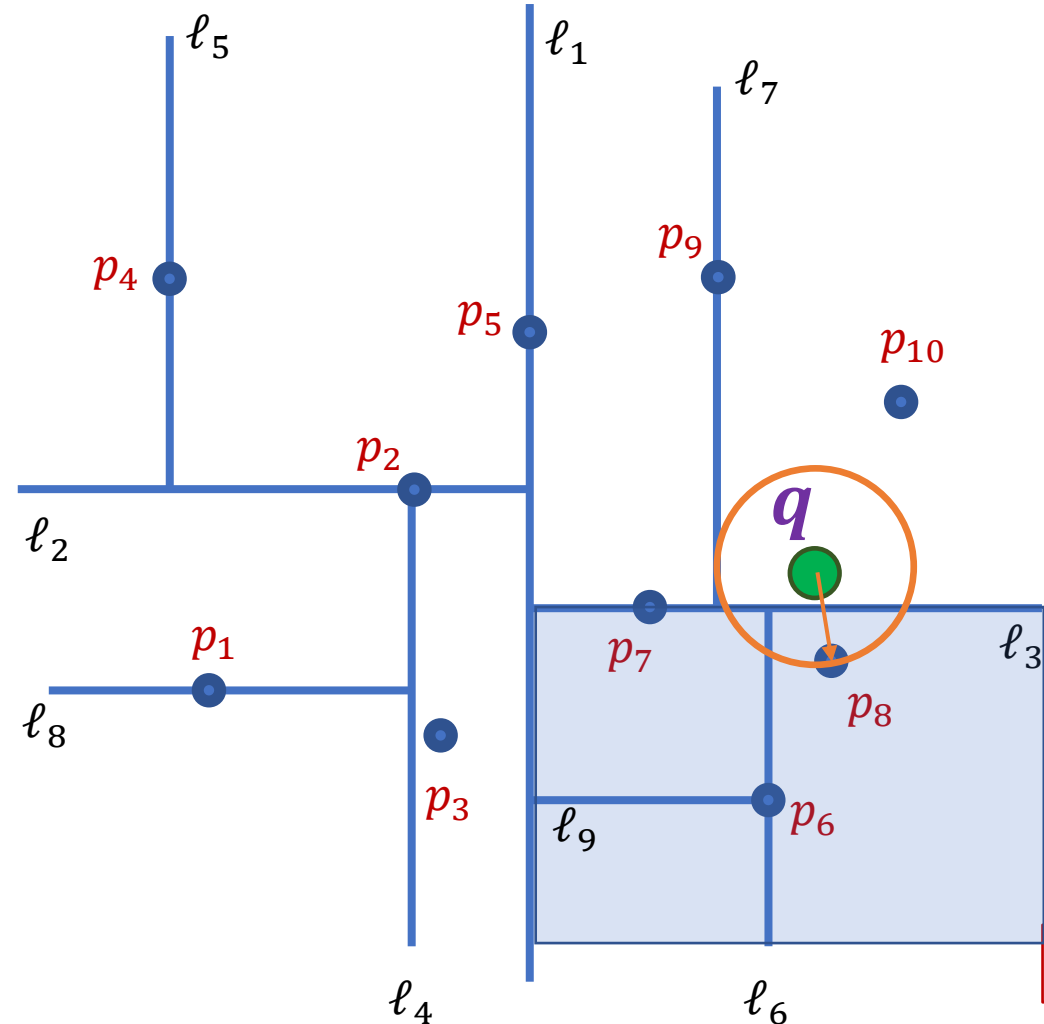


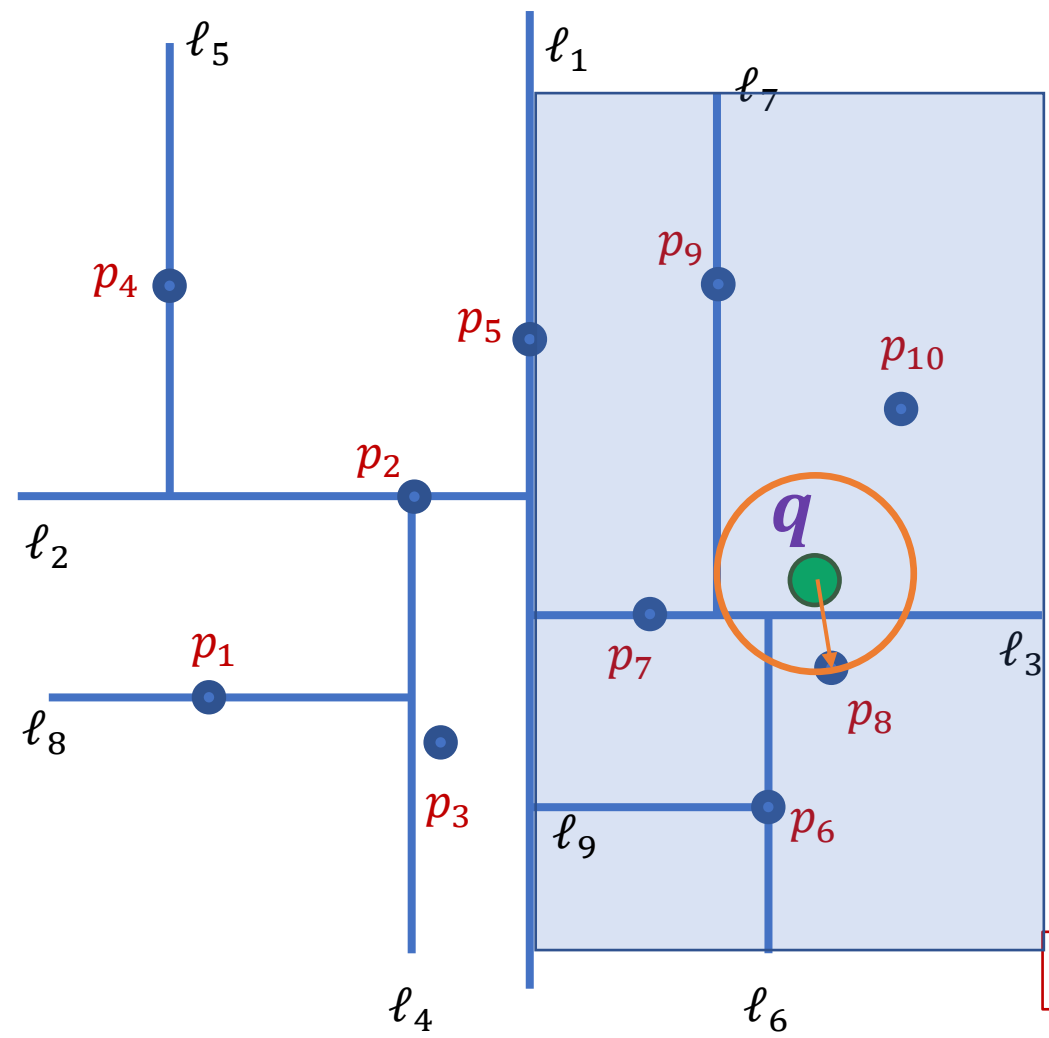
# Updating the candidate to $p_8$

Current candidate:  $p_8$

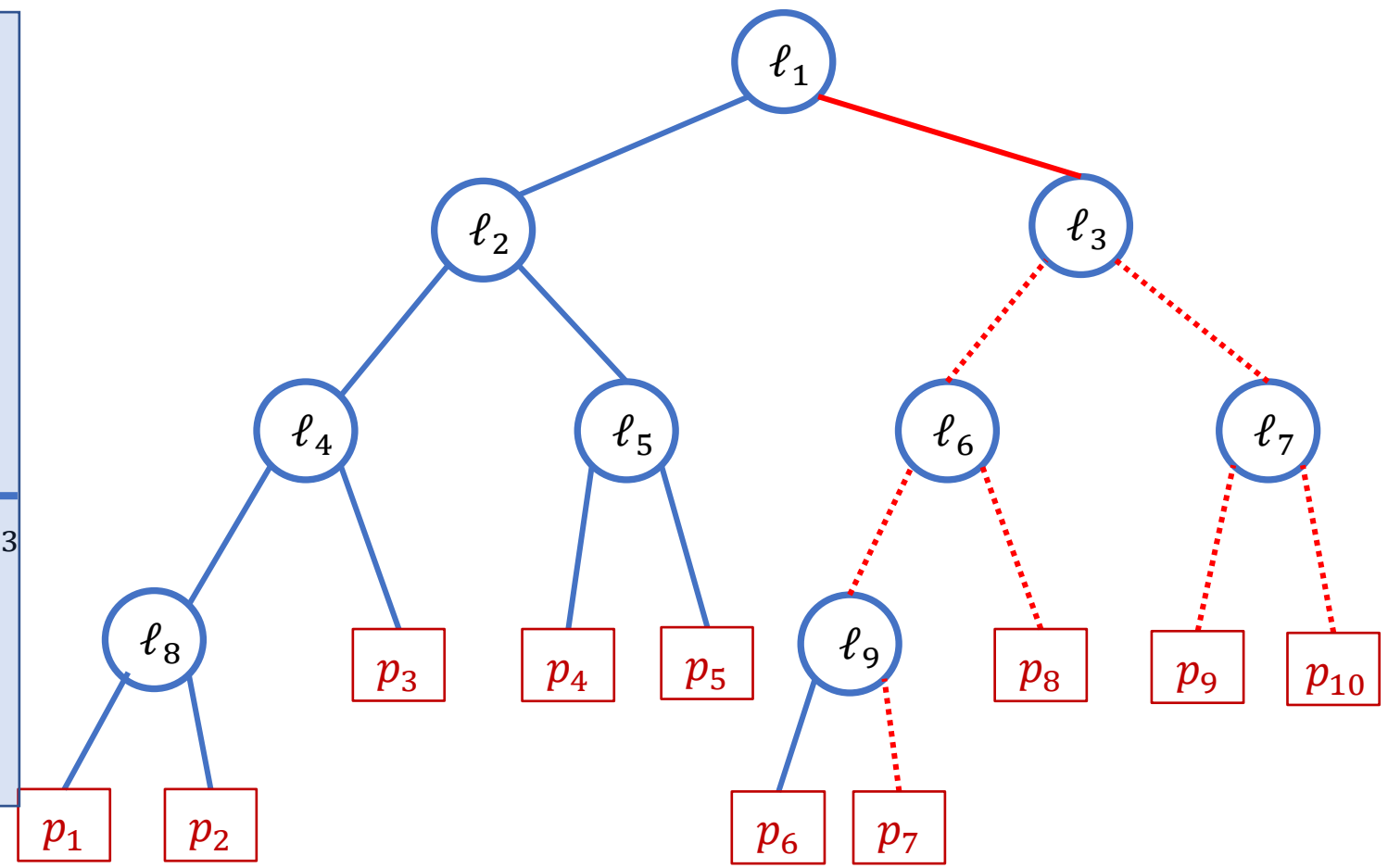


Current candidate:  $p_8$

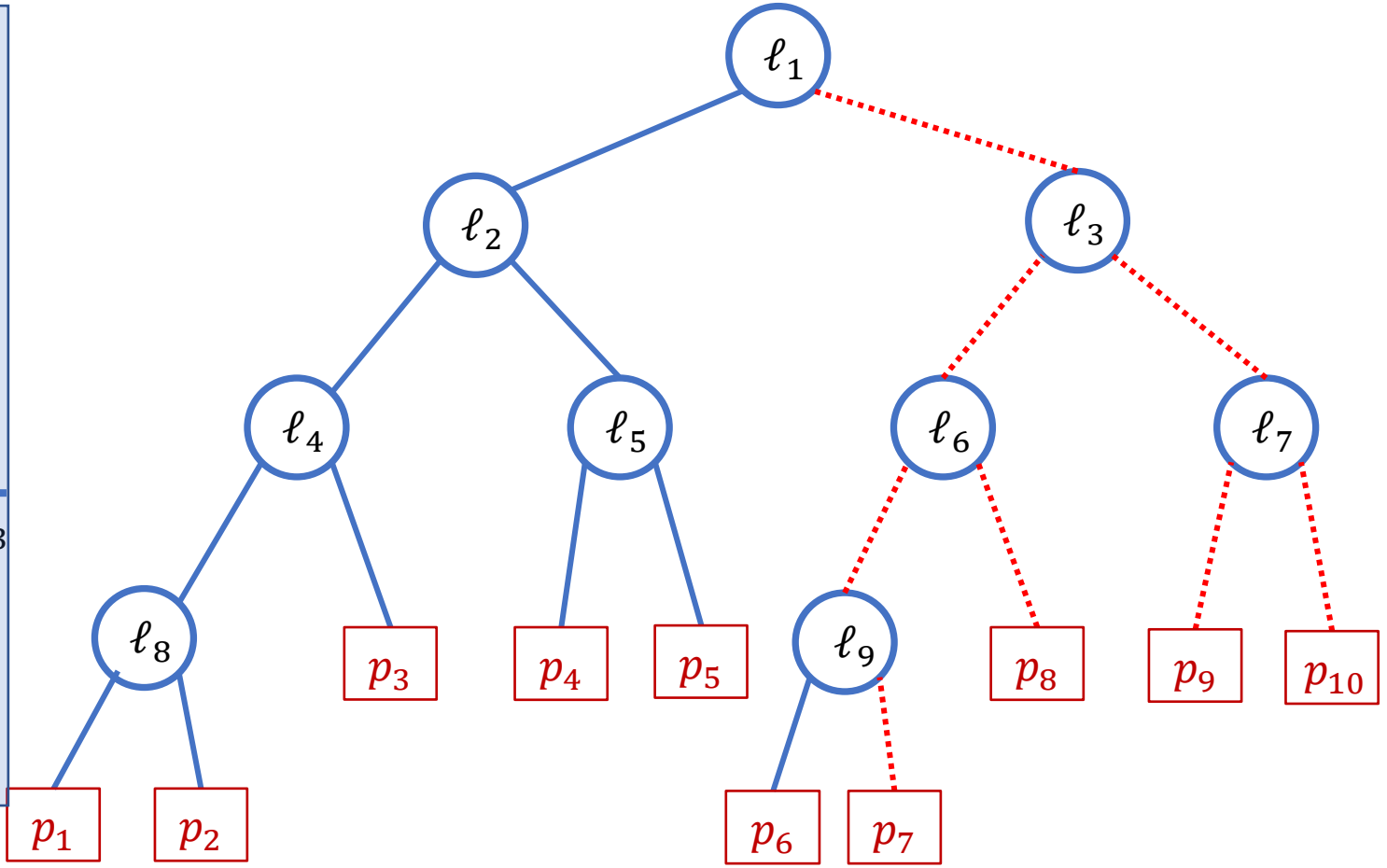
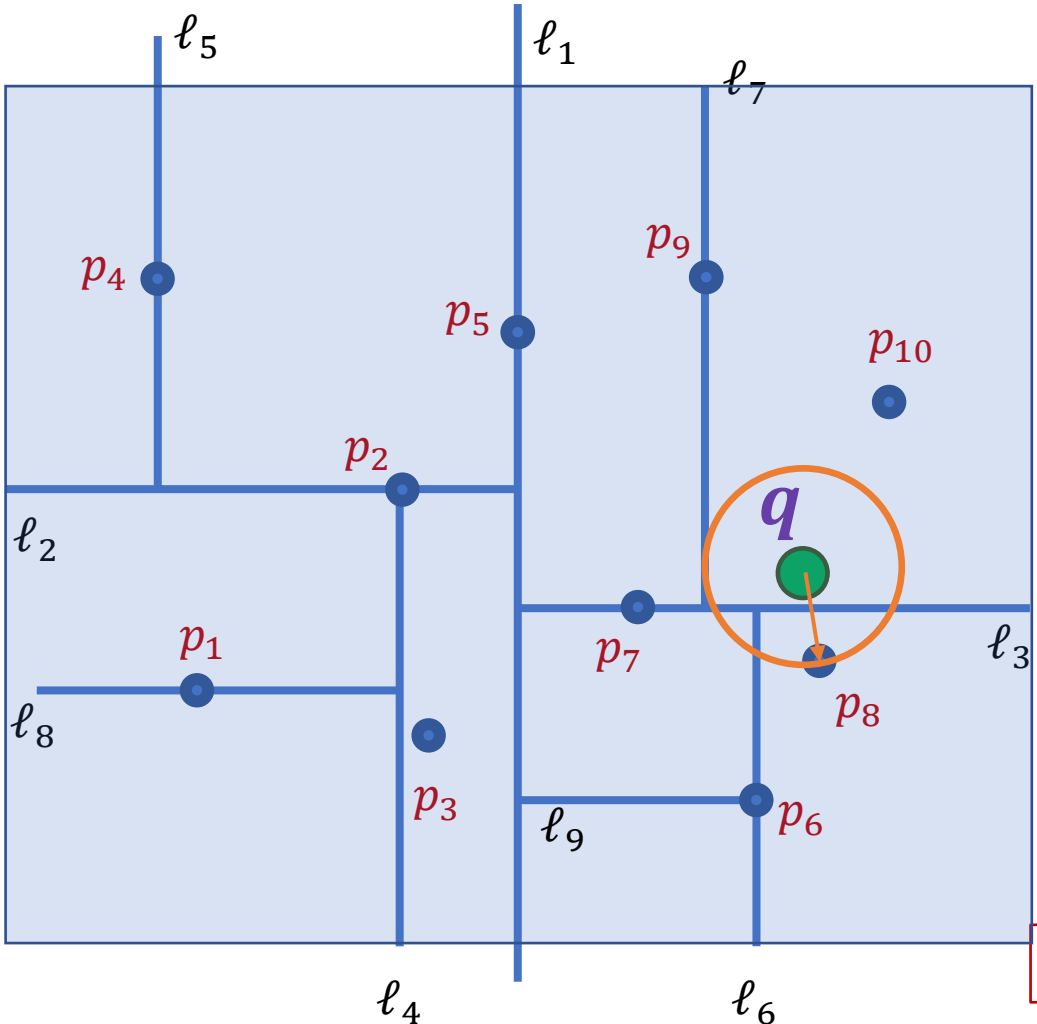




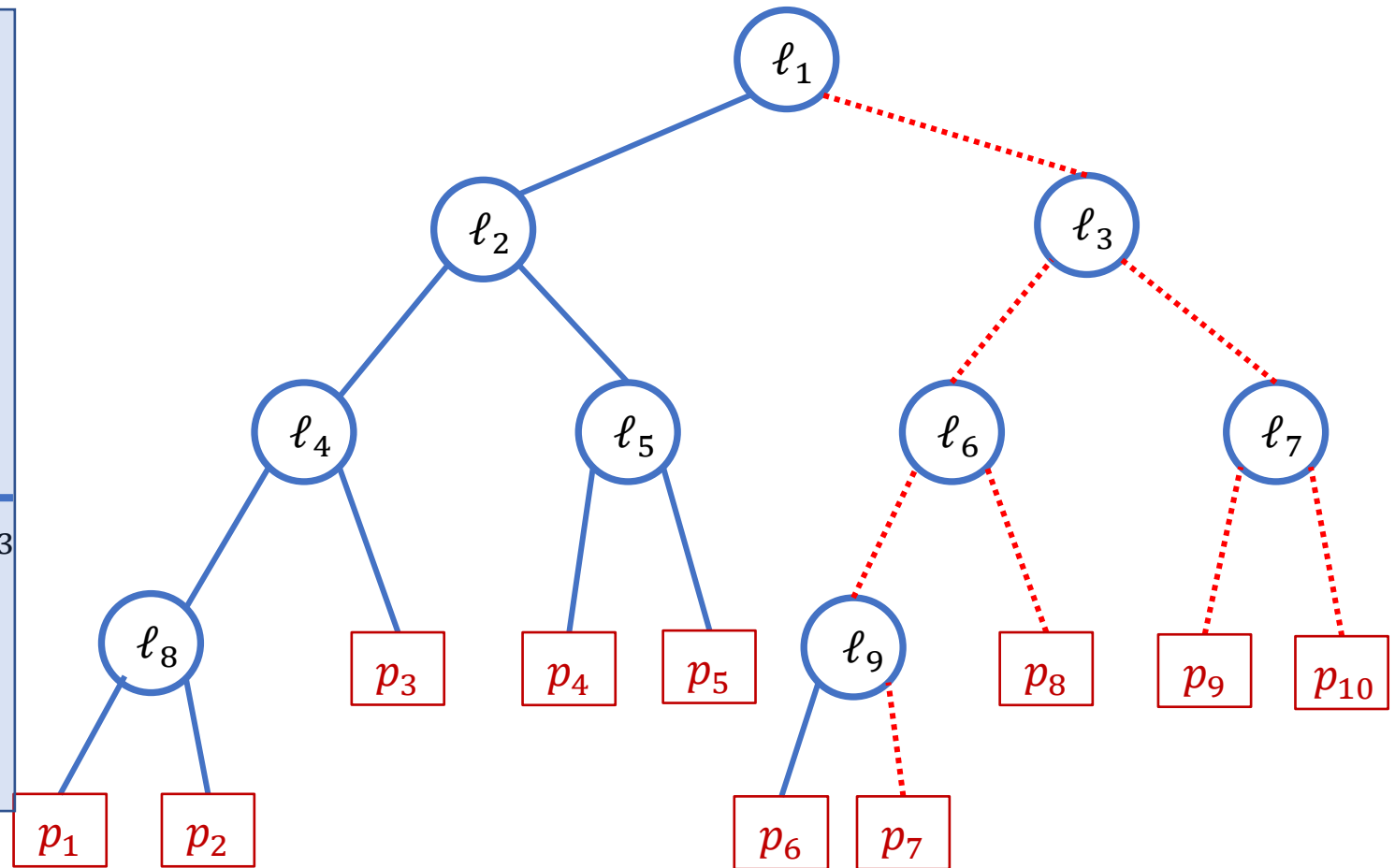
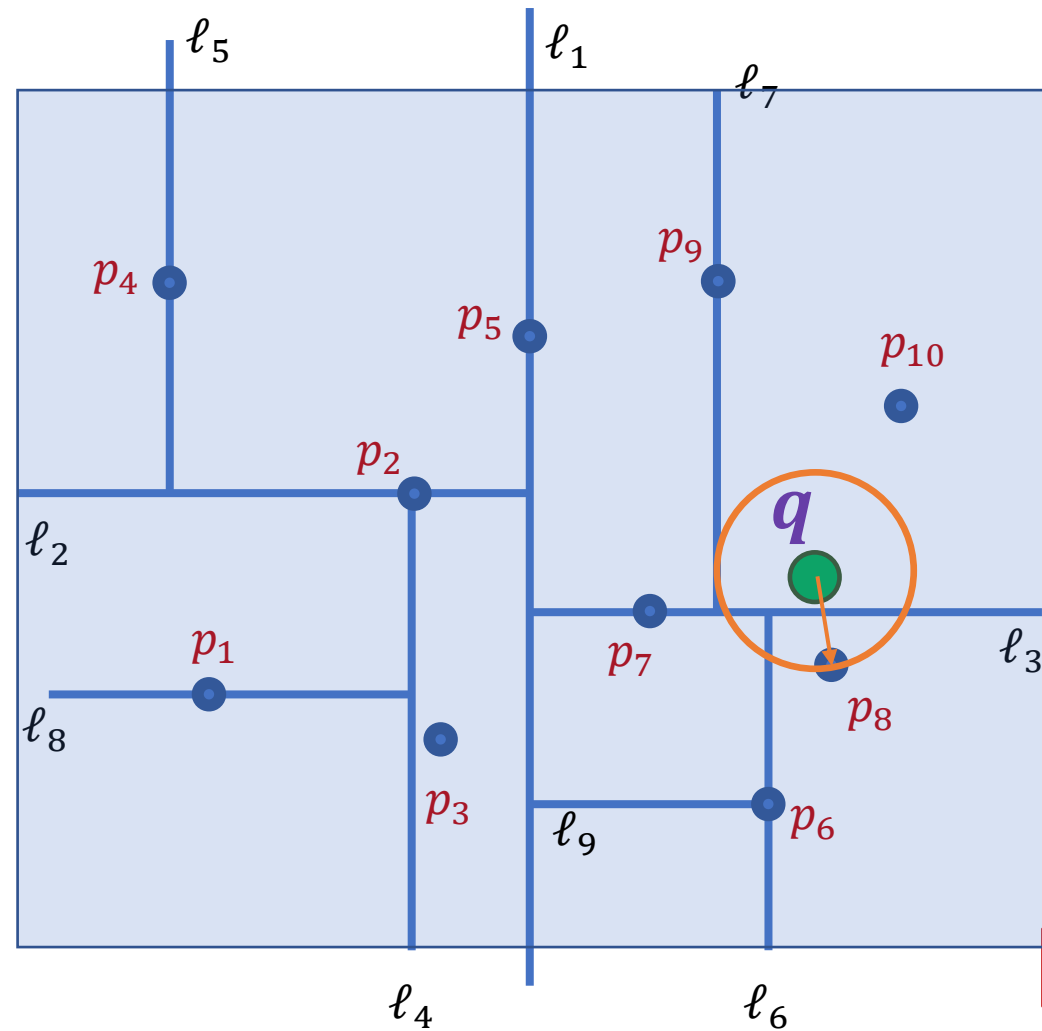
Current candidate:  $p_8$



Current candidate:  $p_8$



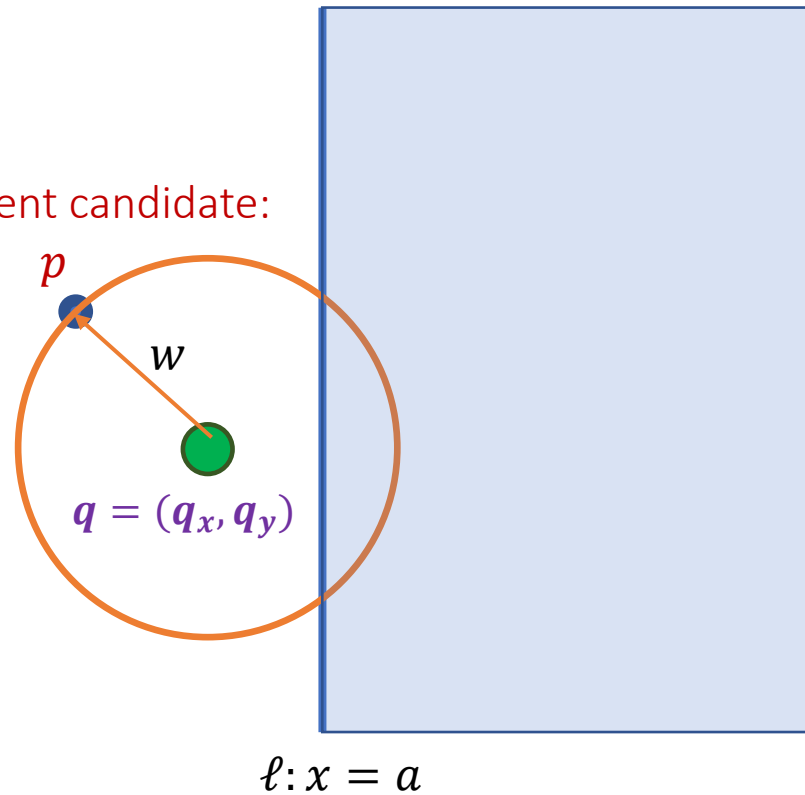
$p_8$  is the nearest neighbor



# Testing for intersection

The box and the circle intersect if:  $q_x + w > a$

The current candidate:



Main is NNS(q,root,null,infinity)

# Nearest Neighbor Search

```
NNS(q: point, n: node, p: point, w: distance) : point {
  if n.left = null then {leaf case}
    if distance(q,n.point) < w then return n.point else return p;
  else
    if w = infinity then
      if q(n.axis) ≤ n.value then
        p := NNS(q,n.left,p,w);
        w := distance(p,q);
        if q(n.axis) + w > n.value then p := NNS(q, n.right, p, w);
      else
        p := NNS(q,n.right,p,w);
        w := distance(p,q);
        if q(n.axis) - w ≤ n.value then p := NNS(q, n.left, p, w);
    else //w is finite//
      if q(n.axis) - w ≤ n.value then
        p := NNS(q, n.left, p, w);
        w := distance(p,q);
        if q(n.axis) + w > n.value then p := NNS(q, n.right, p, w);
    return p
}
```

Taken from <https://courses.cs.washington.edu/courses/cse373/02au/lectures/lecture22l.pdf>

# Complexity

- For a “nicely spread” point set, NN-search on Kd-trees runs in  $O(d \log n)$  average time
- In the worst case the query complexity is  $O(dn)$
- Space complexity is  $O(dn)$