4.3 Redundancy of Prefix Codes

The Entropy Bound Revisited

The Entropy Bound Revisited

- In Section 4.1, we have proved the entropy bound for a D-ary uniquely decodable code, i.e.,

$$L \geq H_D(X).$$

The Entropy Bound Revisited

- In Section 4.1, we have proved the entropy bound for a D-ary uniquely decodable code, i.e.,

$$L \geq H_D(X).$$

- We will present an alternative proof specifically for prefix codes which offers much insight into the redundancy of such codes.

A *D*-ary Code Tree

A *D*-ary Code Tree

- Let X be a source random variable with probability distribution

  $$\{p_1, p_2, \cdots, p_m\},$$

  where $m \geq 2$.

A *D*-ary Code Tree

- Let X be a source random variable with probability distribution

  $$\{p_1, p_2, \cdots, p_m\},$$

  where $m \geq 2$.

- A D-ary prefix code for X can be represented by a D-ary code tree with m leaves, where each leaf corresponds to a codeword.

A *D*-ary Code Tree

- Let X be a source random variable with probability distribution

$$\{p_1, p_2, \cdots, p_m\},$$

  where $m \geq 2$.

- A D-ary prefix code for X can be represented by a D-ary code tree with m leaves, where each leaf corresponds to a codeword.

- Denote the leaf corresponding to p_i by c_i and the order of c_i by l_i.

A *D*-ary Code Tree

- Let X be a source random variable with probability distribution
$$\{p_1, p_2, \cdots, p_m\},$$
where $m \geq 2$.
- A D-ary prefix code for X can be represented by a D-ary code tree with m leaves, where each leaf corresponds to a codeword.
- Denote the leaf corresponding to p_i by c_i and the order of c_i by l_i.
- Let the alphabet be
$$\{0, 1, \cdots, D-1\}.$$

A *D*-ary Code Tree

- Let X be a source random variable with probability distribution
$$\{p_1, p_2, \cdots, p_m\},$$
where $m \geq 2$.

- A D-ary prefix code for X can be represented by a D-ary code tree with m leaves, where each leaf corresponds to a codeword.

- Denote the leaf corresponding to p_i by c_i and the order of c_i by l_i.

- Let the alphabet be
$$\{0, 1, \cdots, D-1\}.$$

- Let $\mathcal{I}$ be the index set of all the internal nodes (including the root) in the code tree.

Reaching Probability

Reaching Probability

- To decode a codeword of a prefix code, we can trace the path specified by the codeword from the root of the code tree until it terminates at the leaf corresponding to that codeword.

Reaching Probability

- To decode a codeword of a prefix code, we can trace the path specified by the codeword from the root of the code tree until it terminates at the leaf corresponding to that codeword.

- Let q_k be the probability of reaching an internal node k during the decoding process.

Reaching Probability

- To decode a codeword of a prefix code, we can trace the path specified by the codeword from the root of the code tree until it terminates at the leaf corresponding to that codeword.

- Let q_k be the probability of reaching an internal node k during the decoding process.

- The probability q_k is called the reaching probability of internal node k.

Reaching Probability

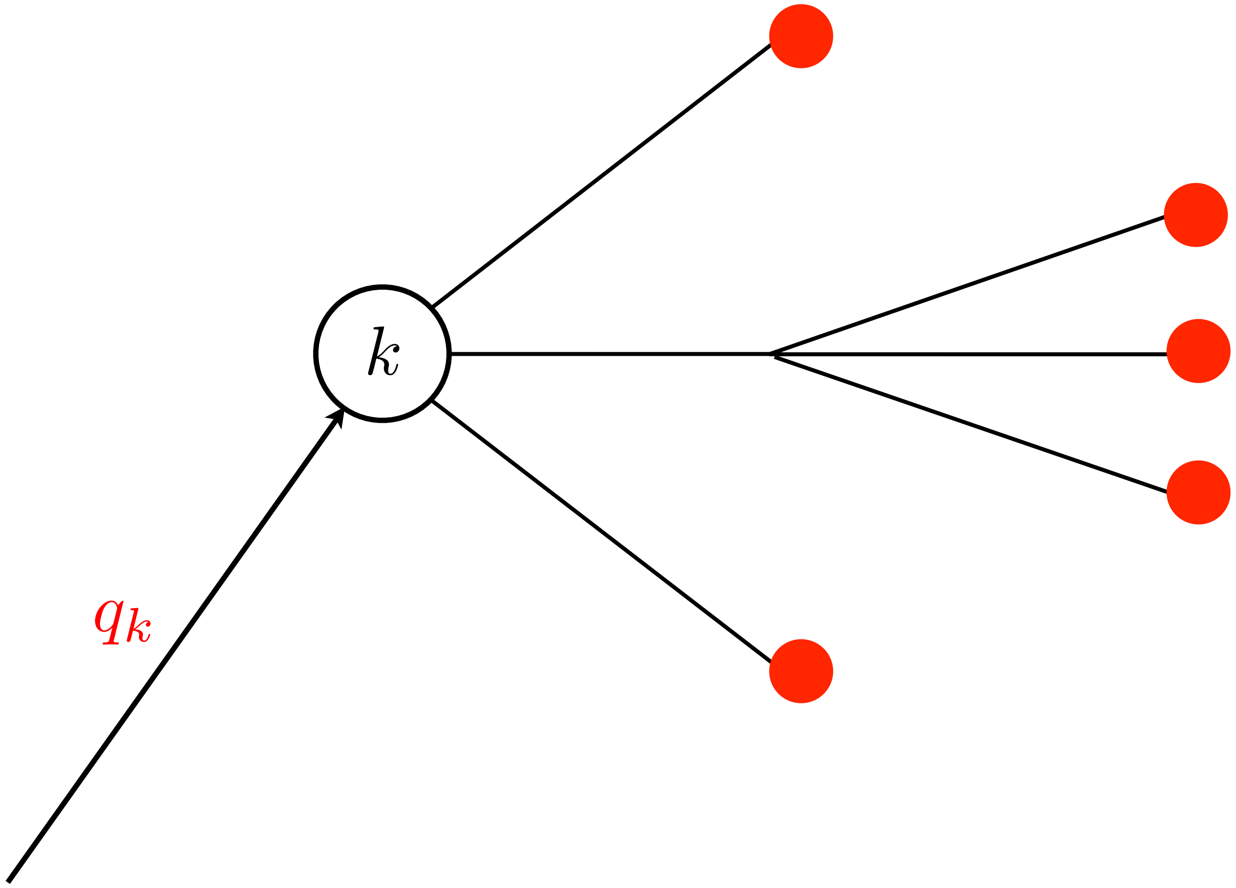
- To decode a codeword of a prefix code, we can trace the path specified by the codeword from the root of the code tree until it terminates at the leaf corresponding to that codeword.

- Let q_k be the probability of reaching an internal node k during the decoding process.

- The probability q_k is called the reaching probability of internal node k.

- q_k is equal to the sum of the probabilities of all the leaves descending from node k.

Reaching Probability



Branching at an Internal Node

Branching at an Internal Node

- Let $\tilde{p}_{k,j}$ be the probability that the jth branch of node k is taken during the decoding process.

Branching at an Internal Node

- Let $\tilde{p}_{k,j}$ be the probability that the jth branch of node k is taken during the decoding process.

- The probabilities $\tilde{p}_{k,j}$, $0 \le j \le D-1$, are called the branching probabilities of node k, and

$$q_k = \sum_j \tilde{p}_{k,j}.$$

Branching at an Internal Node

- Let $\tilde{p}_{k,j}$ be the probability that the jth branch of node k is taken during the decoding process.

- The probabilities $\tilde{p}_{k,j}, 0 \leq j \leq D-1$, are called the branching probabilities of node k, and

$$q_k = \sum_j \tilde{p}_{k,j}.$$

- Once node k is reached, the conditional branching distribution is

$$\left\{ \frac{\tilde{p}_{k,0}}{q_k}, \frac{\tilde{p}_{k,1}}{q_k}, \cdots, \frac{\tilde{p}_{k,D-1}}{q_k} \right\}.$$

Branching at an Internal Node

- Let $\tilde{p}_{k,j}$ be the probability that the jth branch of node k is taken during the decoding process.

- The probabilities $\tilde{p}_{k,j}, 0 \le j \le D-1$, are called the branching probabilities of node k, and

$$q_k = \sum_j \tilde{p}_{k,j}.$$

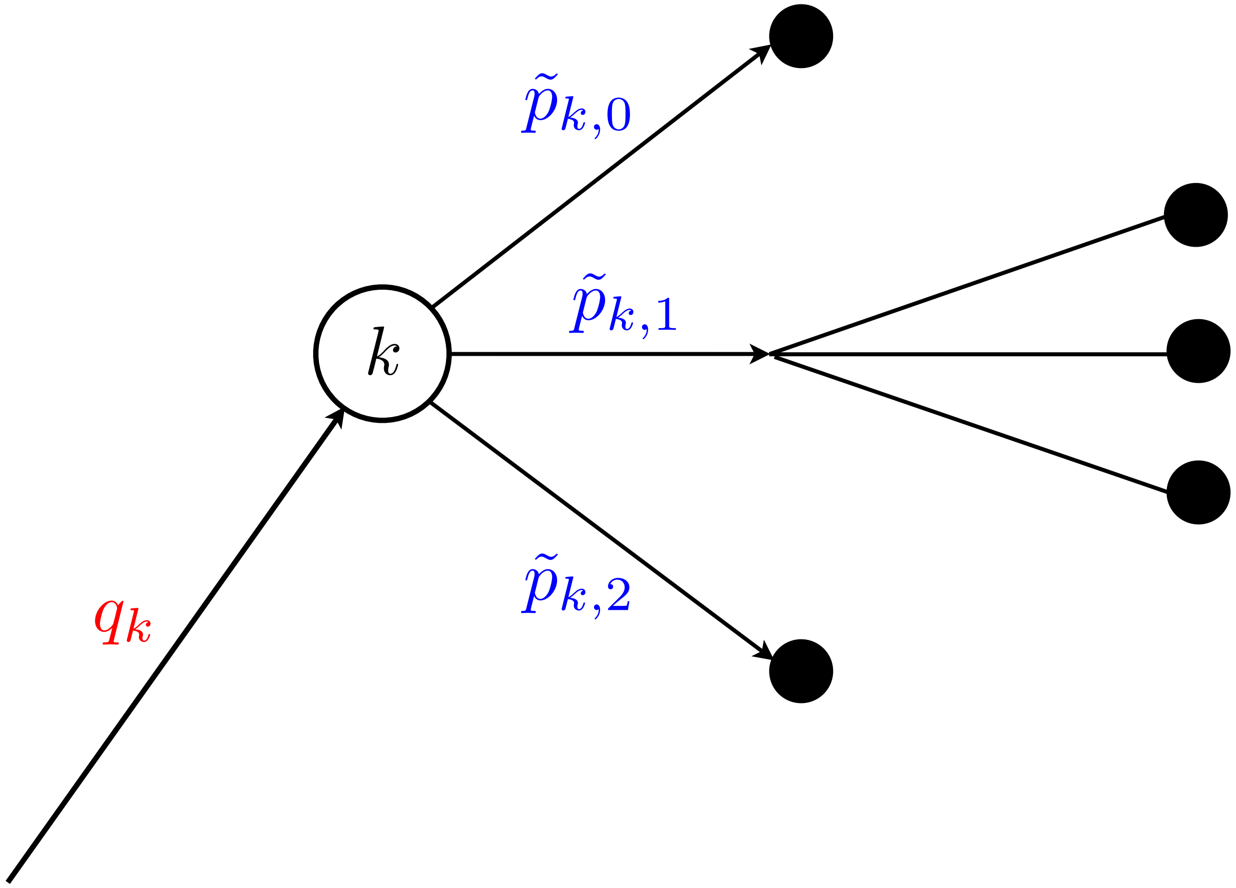
- Once node k is reached, the conditional branching distribution is

$$\left\{ \frac{\tilde{p}_{k,0}}{q_k}, \frac{\tilde{p}_{k,1}}{q_k}, \cdots, \frac{\tilde{p}_{k,D-1}}{q_k} \right\}.$$

- Then define the conditional entropy of node k by

$$h_k = H_D\left(\left\{ \frac{\tilde{p}_{k,0}}{q_k}, \frac{\tilde{p}_{k,1}}{q_k}, \cdots, \frac{\tilde{p}_{k,D-1}}{q_k} \right\}\right) \le \log_D D = 1.$$

Conditional Branching Distribution



Lemma 4.19 $H_D(X) = \sum_{k \in \mathcal{I}} q_k \, h_k$.

Lemma 4.19 $H_D(X) = \sum_{k \in \mathcal{I}} q_k \, h_k$.

q_k: reaching probability of internal node k

Lemma 4.19 $H_D(X) = \sum_{k \in \mathcal{I}} q_k \, h_k$.

q_k: reaching probability of internal node k

h_k: conditional entropy of internal node k

Lemma 4.19 $H_D(X) = \sum_{k \in \mathcal{I}} q_k h_k$.

Proof

Lemma 4.19 $H_D(X) = \sum_{k \in \mathcal{I}} q_k h_k$.

Proof

1. We prove the lemma by induction on the number of internal nodes of the code tree.

Lemma 4.19 $H_D(X) = \sum_{k \in \mathcal{I}} q_k h_k$.

Proof

1. We prove the lemma by induction on the number of internal nodes of the code tree.

2. If there is only one internal node, it must be the root of the tree. Then the lemma is trivially true upon observing that the reaching probability of the root is equal to 1.

Lemma 4.19 $H_D(X) = \sum_{k \in \mathcal{I}} q_k h_k$.

Proof

1. We prove the lemma by induction on the number of internal nodes of the code tree.

2. If there is only one internal node, it must be the root of the tree. Then the lemma is trivially true upon observing that the reaching probability of the root is equal to 1.

Lemma 4.19 $H_D(X) = \sum_{k \in \mathcal{I}} q_k h_k$.

Proof

1. We prove the lemma by induction on the number of internal nodes of the code tree.

2. If there is only one internal node, it must be the root of the tree. Then the lemma is trivially true upon observing that the reaching probability of the root is equal to 1.

- Recall that the source distribution is

$$\{p_1, p_2, \cdots, p_m\}.$$

Lemma 4.19 $H_D(X) = \sum_{k \in \mathcal{I}} q_k h_k$.

Proof

1. We prove the lemma by induction on the number of internal nodes of the code tree.

2. If there is only one internal node, it must be the root of the tree. Then the lemma is trivially true upon observing that the reaching probability of the root is equal to 1.

- Recall that the source distribution is

$$\{p_1, p_2, \cdots, p_m\}.$$

- $m \leq D$, otherwise the code tree must have at least 2 internal nodes.

Lemma 4.19 $H_D(X) = \sum_{k \in \mathcal{I}} q_k h_k$.

Proof

1. We prove the lemma by induction on the number of internal nodes of the code tree.

2. If there is only one internal node, it must be the root of the tree. Then the lemma is trivially true upon observing that the reaching probability of the root is equal to 1.

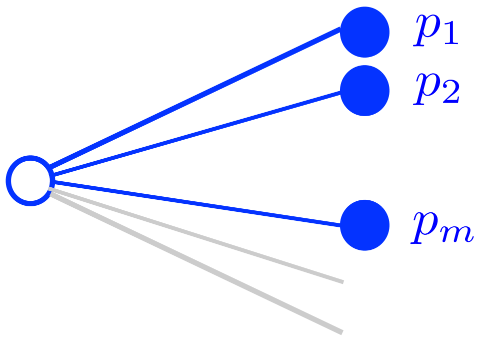
- Recall that the source distribution is
$$\{p_1, p_2, \cdots, p_m\}.$$
- $m \leq D$, otherwise the code tree must have at least 2 internal nodes.
- All the codewords have length equal to 1.

Lemma 4.19 $H_D(X) = \sum_{k \in \mathcal{I}} q_k h_k$.

Proof

1. We prove the lemma by induction on the number of internal nodes of the code tree.

2. If there is only one internal node, it must be the root of the tree. Then the lemma is trivially true upon observing that the reaching probability of the root is equal to 1.



- Recall that the source distribution is

$$\{p_1, p_2, \cdots, p_m\}.$$

- $m \le D$, otherwise the code tree must have at least 2 internal nodes.

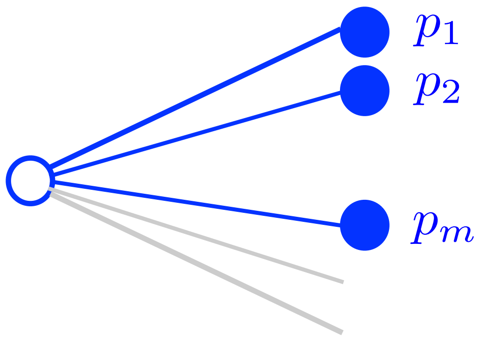
- All the codewords have length equal to 1.

Lemma 4.19 $H_D(X) = \sum_{k \in \mathcal{I}} q_k h_k$.

Proof

1. We prove the lemma by induction on the number of internal nodes of the code tree.

2. If there is only one internal node, it must be the root of the tree. Then the lemma is trivially true upon observing that the reaching probability of the root is equal to 1.



- Recall that the source distribution is

$$\{p_1, p_2, \cdots, p_m\}.$$

- $m \le D$, otherwise the code tree must have at least 2 internal nodes.

- All the codewords have length equal to 1.

- Since $q_{root} = 1$, it is easy to see that the conditional branching distribution of the root is

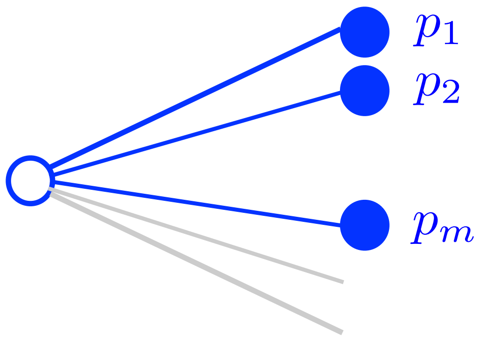
$$\{p_1, p_2, \cdots, p_m, 0, \cdots, 0\}.$$

Lemma 4.19 $H_D(X) = \sum_{k \in \mathcal{I}} q_k h_k$.

Proof

1. We prove the lemma by induction on the number of internal nodes of the code tree.

2. If there is only one internal node, it must be the root of the tree. Then the lemma is trivially true upon observing that the reaching probability of the root is equal to 1.



- Recall that the source distribution is

$$\{p_1, p_2, \cdots, p_m\}.$$

- $m \leq D$, otherwise the code tree must have at least 2 internal nodes.

- All the codewords have length equal to 1.

- Since $q_{root} = 1$, it is easy to see that the conditional branching distribution of the root is

$$\{p_1, p_2, \cdots, p_m, 0, \cdots, 0\}.$$

- Then

$$h_{root} = H_D(\{p_1, p_2, \cdots, p_m, 0, \cdots, 0\}),$$

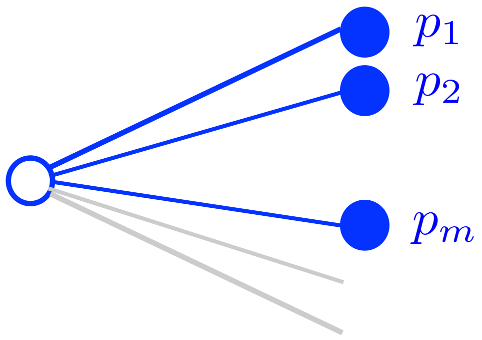
which is equal to $H_D(X)$.

Lemma 4.19 $H_D(X) = \sum_{k \in \mathcal{I}} q_k h_k$.

Proof

1. We prove the lemma by induction on the number of internal nodes of the code tree.

2. If there is only one internal node, it must be the root of the tree. Then the lemma is trivially true upon observing that the reaching probability of the root is equal to 1.



- Recall that the source distribution is
$$\{p_1, p_2, \cdots, p_m\}.$$

- $m \leq D$, otherwise the code tree must have at least 2 internal nodes.

- All the codewords have length equal to 1.

- Since $q_{root} = 1$, it is easy to see that the conditional branching distribution of the root is
$$\{p_1, p_2, \cdots, p_m, 0, \cdots, 0\}.$$

- Then
$$h_{root} = H_D(\{p_1, p_2, \cdots, p_m, 0, \cdots, 0\}),$$
which is equal to $H_D(X)$.

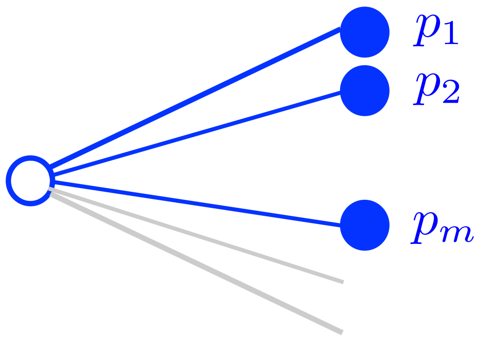
- It follows that
$$\sum_{k \in \mathcal{I}} q_k h_k = q_{root} \cdot h_{root} = 1 \cdot H_D(X).$$

Lemma 4.19 $H_D(X) = \sum_{k \in \mathcal{I}} q_k h_k$.

Proof

1. We prove the lemma by induction on the number of internal nodes of the code tree.

2. If there is only one internal node, it must be the root of the tree. Then the lemma is trivially true upon observing that the reaching probability of the root is equal to 1.



- Recall that the source distribution is

$$\{p_1, p_2, \cdots, p_m\}.$$

- $m \leq D$, otherwise the code tree must have at least 2 internal nodes.

- All the codewords have length equal to 1.

- Since $q_{root} = 1$, it is easy to see that the conditional branching distribution of the root is

$$\{p_1, p_2, \cdots, p_m, 0, \cdots, 0\}.$$

- Then

$$h_{root} = H_D(\{p_1, p_2, \cdots, p_m, 0, \cdots, 0\}),$$

which is equal to $H_D(X)$.

- It follows that

$$\sum_{k \in \mathcal{I}} q_k h_k = q_{root} \cdot h_{root} = 1 \cdot H_D(X).$$

Lemma 4.19 $H_D(X) = \sum_{k \in \mathcal{I}} q_k h_k$.

Proof

1. We prove the lemma by induction on the number of internal nodes of the code tree.

2. If there is only one internal node, it must be the root of the tree. Then the lemma is trivially true upon observing that the reaching probability of the root is equal to 1.

Lemma 4.19 $H_D(X) = \sum_{k\in\mathcal{I}} q_k h_k$.

Proof

1. We prove the lemma by induction on the number of internal nodes of the code tree.

2. If there is only one internal node, it must be the root of the tree. Then the lemma is trivially true upon observing that the reaching probability of the root is equal to 1.

3. Assume the lemma is true for all code trees with n internal nodes, and consider a code tree with $n + 1$ internal nodes.

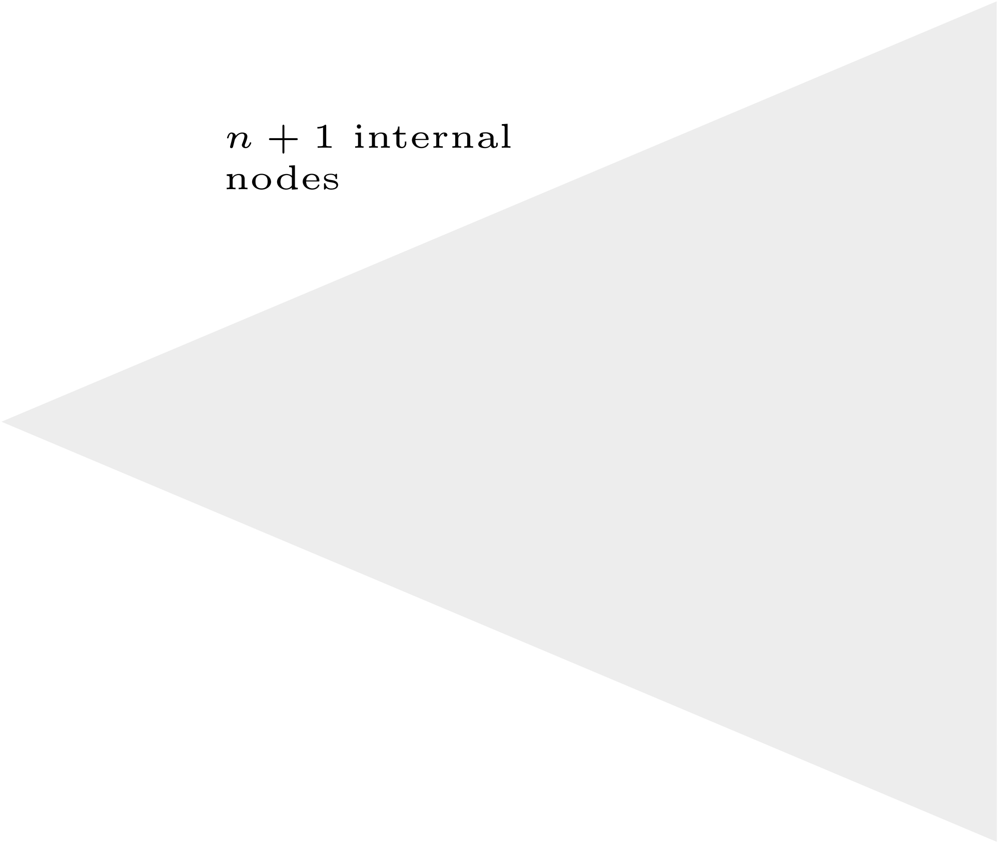
Lemma 4.19 $H_D(X) = \sum_{k \in \mathcal{I}} q_k h_k$.

Proof

1. We prove the lemma by induction on the number of internal nodes of the code tree.

2. If there is only one internal node, it must be the root of the tree. Then the lemma is trivially true upon observing that the reaching probability of the root is equal to 1.

3. Assume the lemma is true for all code trees with n internal nodes, and consider a code tree with $n + 1$ internal nodes.



Lemma 4.19 $H_D(X) = \sum_{k \in \mathcal{I}} q_k h_k$.

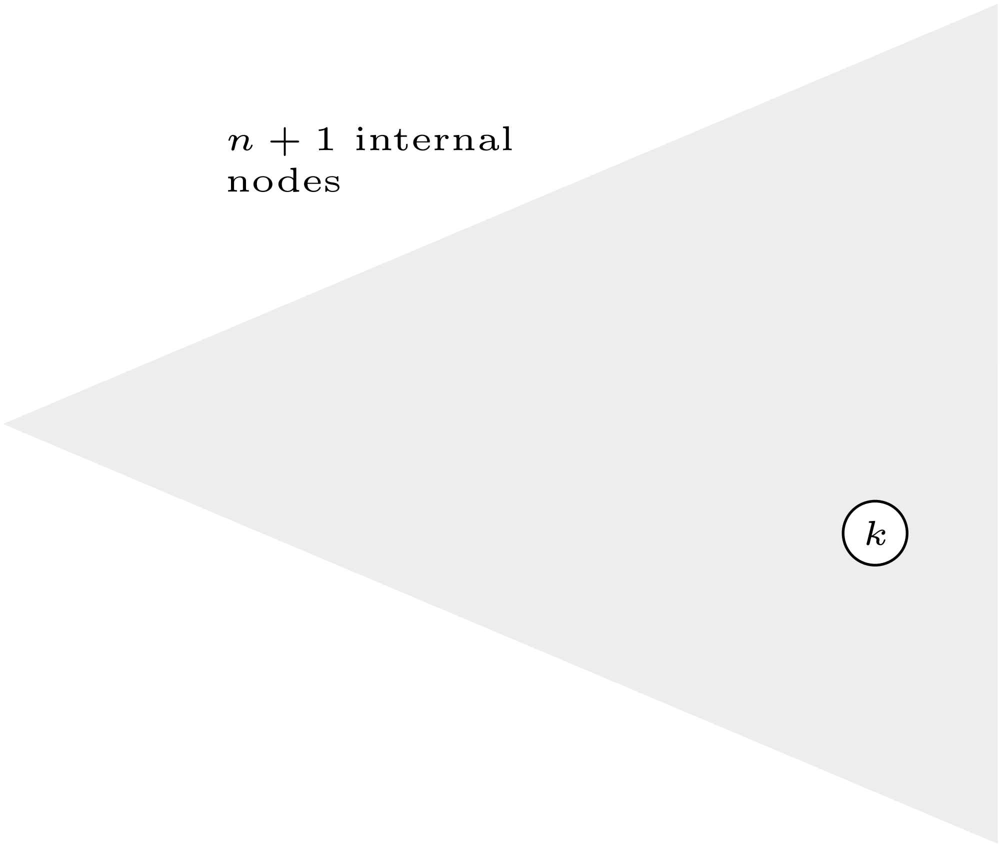
Proof

1. We prove the lemma by induction on the number of internal nodes of the code tree.

2. If there is only one internal node, it must be the root of the tree. Then the lemma is trivially true upon observing that the reaching probability of the root is equal to 1.

3. Assume the lemma is true for all code trees with n internal nodes, and consider a code tree with $n+1$ internal nodes.

4. Let k be an internal node such that k is the parent of a leaf c with maximum order.



$n + 1$ internal nodes

Lemma 4.19 $H_D(X) = \sum_{k \in \mathcal{I}} q_k h_k$.

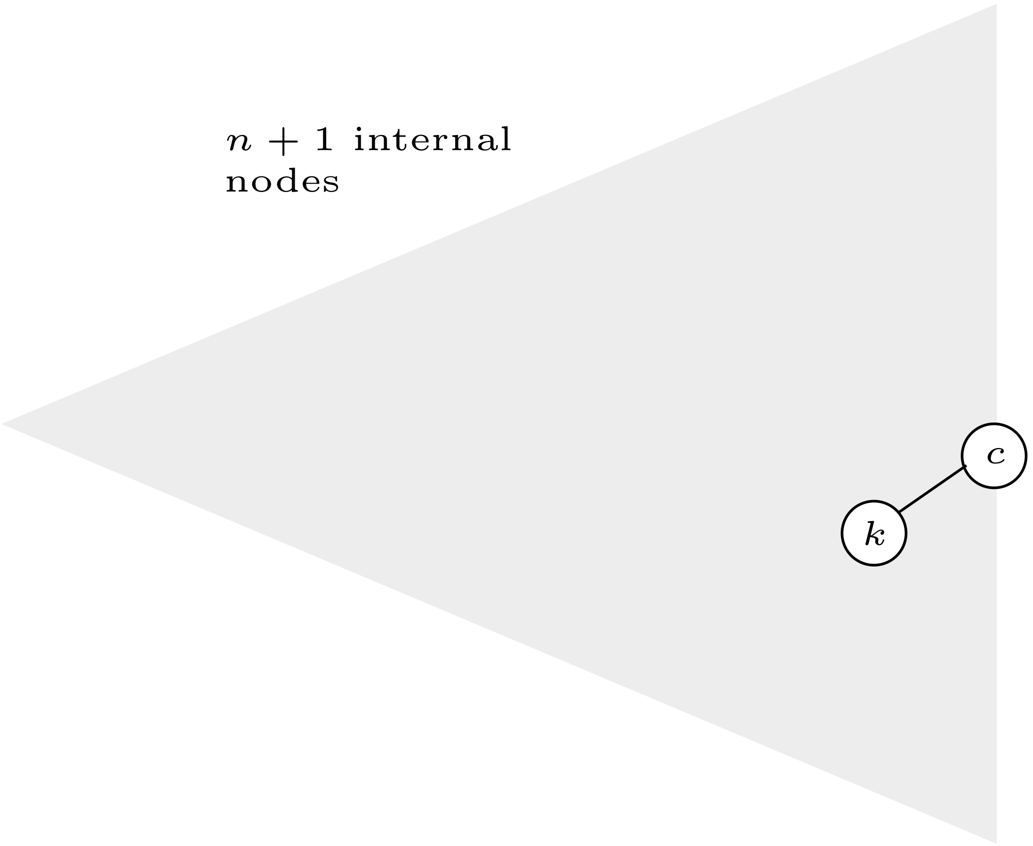
Proof

1. We prove the lemma by induction on the number of internal nodes of the code tree.

2. If there is only one internal node, it must be the root of the tree. Then the lemma is trivially true upon observing that the reaching probability of the root is equal to 1.

3. Assume the lemma is true for all code trees with n internal nodes, and consider a code tree with $n+1$ internal nodes.

4. Let k be an internal node such that k is the parent of a leaf c with maximum order.



Lemma 4.19 $H_D(X) = \sum_{k \in \mathcal{I}} q_k h_k$.

Proof

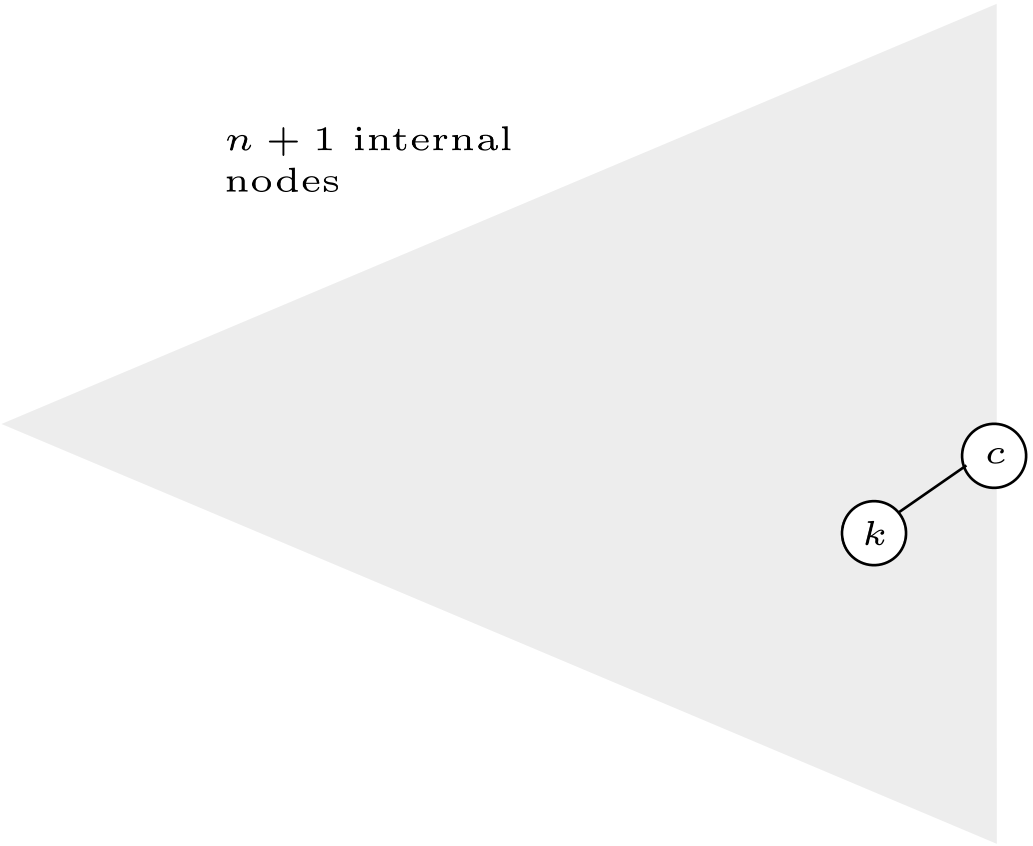
1. We prove the lemma by induction on the number of internal nodes of the code tree.

2. If there is only one internal node, it must be the root of the tree. Then the lemma is trivially true upon observing that the reaching probability of the root is equal to 1.

3. Assume the lemma is true for all code trees with n internal nodes, and consider a code tree with $n + 1$ internal nodes.

4. Let k be an internal node such that k is the parent of a leaf c with maximum order.

5. Each sibling of c may or may not be a leaf. If it is not a leaf, then it cannot be the ascendent of another leaf because we assume that c is a leaf with maximum order.



Lemma 4.19 $H_D(X) = \sum_{k\in\mathcal{I}} q_k h_k$.

Proof

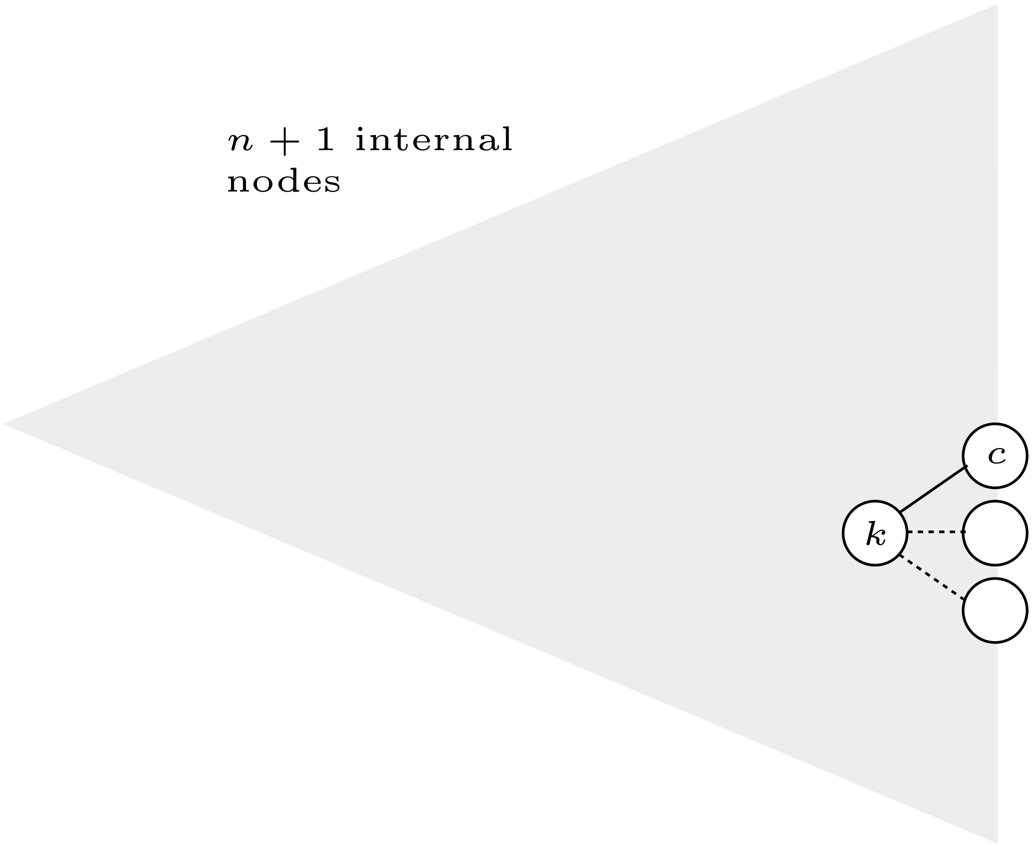
1. We prove the lemma by induction on the number of internal nodes of the code tree.

2. If there is only one internal node, it must be the root of the tree. Then the lemma is trivially true upon observing that the reaching probability of the root is equal to 1.

3. Assume the lemma is true for all code trees with n internal nodes, and consider a code tree with $n+1$ internal nodes.

4. Let k be an internal node such that k is the parent of a leaf c with maximum order.

5. Each sibling of c may or may not be a leaf. If it is not a leaf, then it cannot be the ascendent of another leaf because we assume that c is a leaf with maximum order.



Lemma 4.19 $H_D(X) = \sum_{k\in\mathcal{I}} q_k h_k$.

Proof

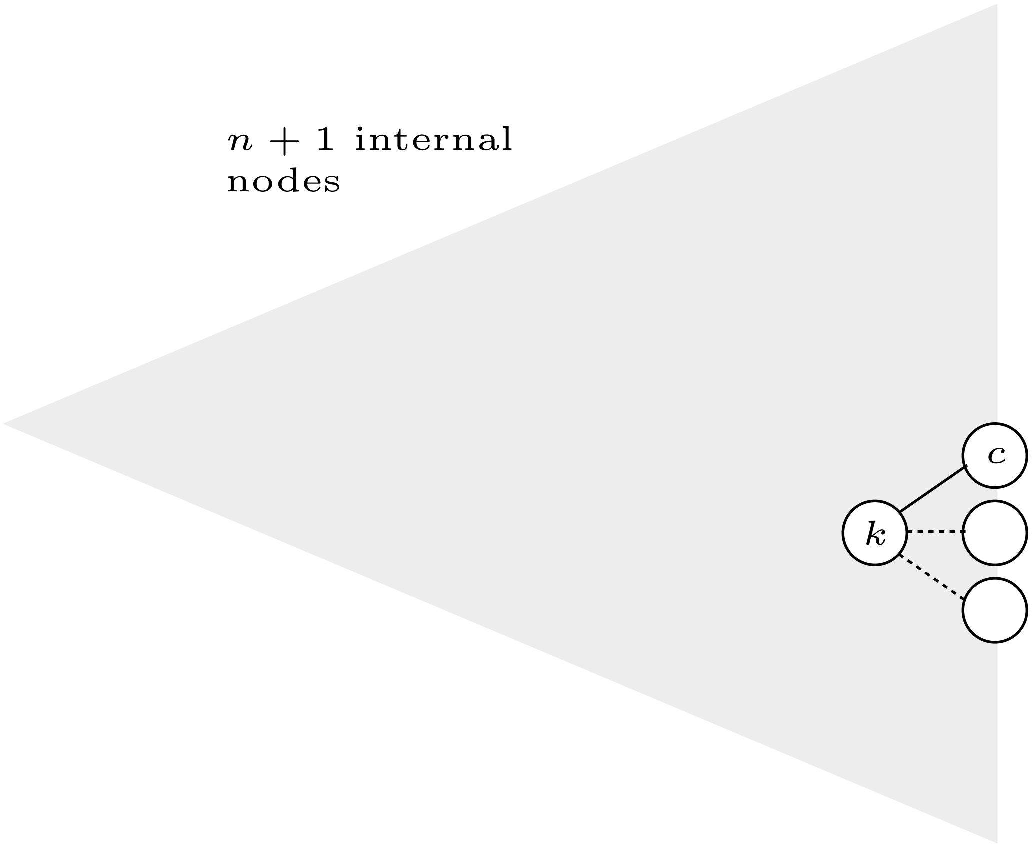
1. We prove the lemma by induction on the number of internal nodes of the code tree.

2. If there is only one internal node, it must be the root of the tree. Then the lemma is trivially true upon observing that the reaching probability of the root is equal to 1.

3. Assume the lemma is true for all code trees with n internal nodes, and consider a code tree with $n+1$ internal nodes.

4. Let k be an internal node such that k is the parent of a leaf c with maximum order.

5. Each sibling of c may or may not be a leaf. If it is not a leaf, then it cannot be the ascendent of another leaf because we assume that c is a leaf with maximum order.



Lemma 4.19 $H_D(X) = \sum_{k \in \mathcal{I}} q_k h_k$.

Proof

1. We prove the lemma by induction on the number of internal nodes of the code tree.

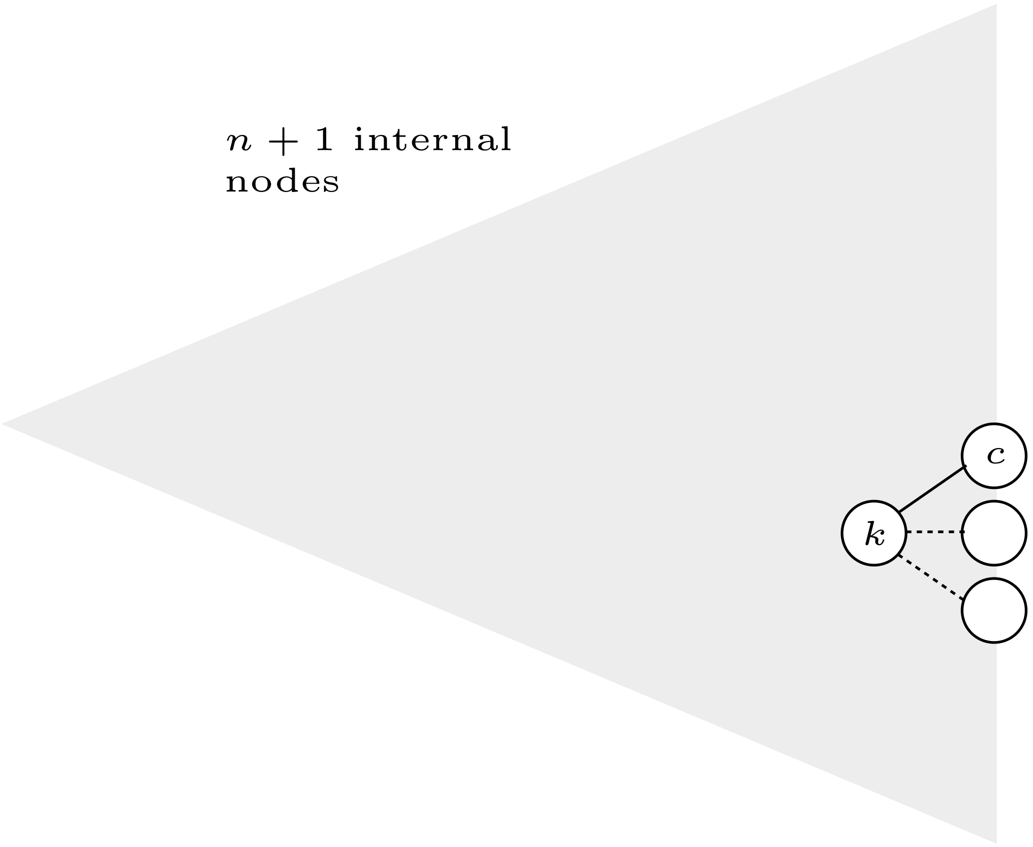
2. If there is only one internal node, it must be the root of the tree. Then the lemma is trivially true upon observing that the reaching probability of the root is equal to 1.

3. Assume the lemma is true for all code trees with n internal nodes, and consider a code tree with $n+1$ internal nodes.

4. Let k be an internal node such that k is the parent of a leaf c with maximum order.

5. Each sibling of c may or may not be a leaf. If it is not a leaf, then it cannot be the ascendent of another leaf because we assume that c is a leaf with maximum order.

6. Now consider revealing the outcome of X in two steps. In the first step, if the outcome of X is not a leaf descending from node k, we identify the outcome exactly, otherwise we identify the outcome to be a child of node k. We call this random variable V.



Lemma 4.19 $H_D(X) = \sum_{k\in\mathcal{I}} q_k h_k$.

Proof

1. We prove the lemma by induction on the number of internal nodes of the code tree.

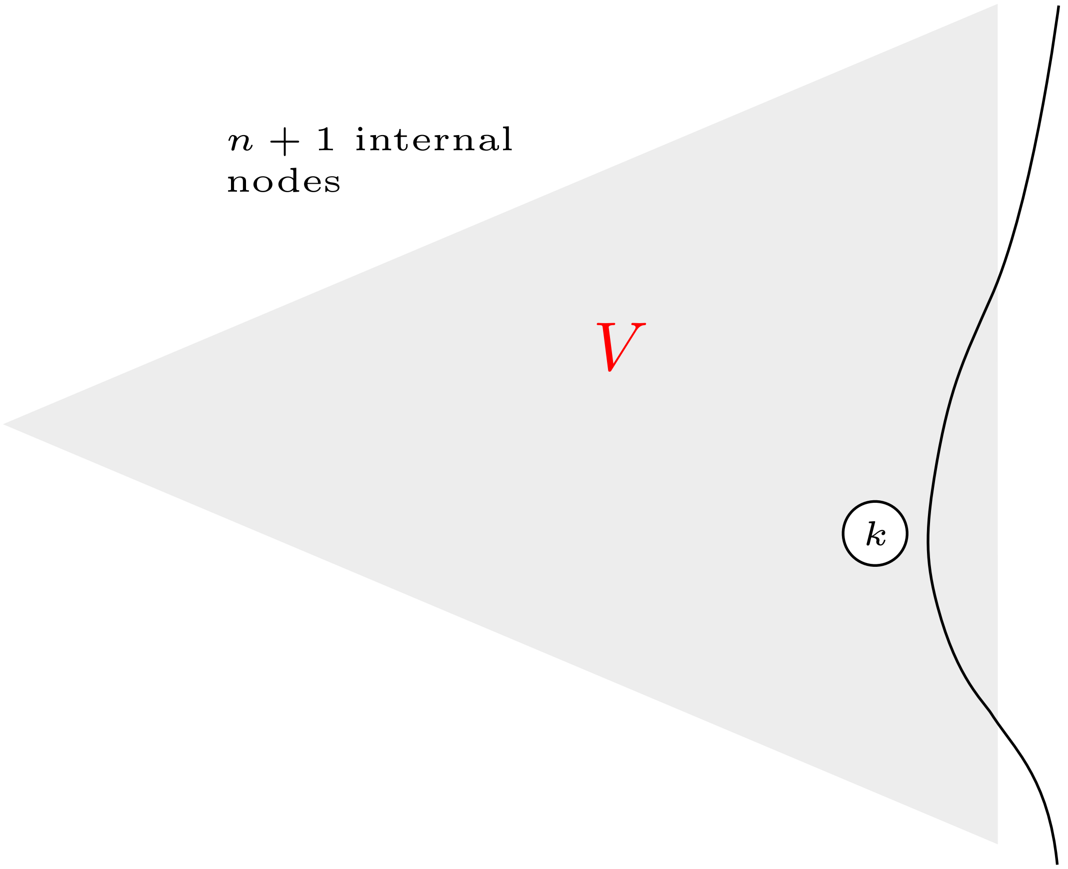
2. If there is only one internal node, it must be the root of the tree. Then the lemma is trivially true upon observing that the reaching probability of the root is equal to 1.

3. Assume the lemma is true for all code trees with n internal nodes, and consider a code tree with $n+1$ internal nodes.

4. Let k be an internal node such that k is the parent of a leaf c with maximum order.

5. Each sibling of c may or may not be a leaf. If it is not a leaf, then it cannot be the ascendent of another leaf because we assume that c is a leaf with maximum order.

6. Now consider revealing the outcome of X in two steps. In the first step, if the outcome of X is not a leaf descending from node k, we identify the outcome exactly, otherwise we identify the outcome to be a child of node k. We call this random variable V.



Lemma 4.19 $H_D(X) = \sum_{k \in \mathcal{I}} q_k h_k$.

Proof

1. We prove the lemma by induction on the number of internal nodes of the code tree.

2. If there is only one internal node, it must be the root of the tree. Then the lemma is trivially true upon observing that the reaching probability of the root is equal to 1.

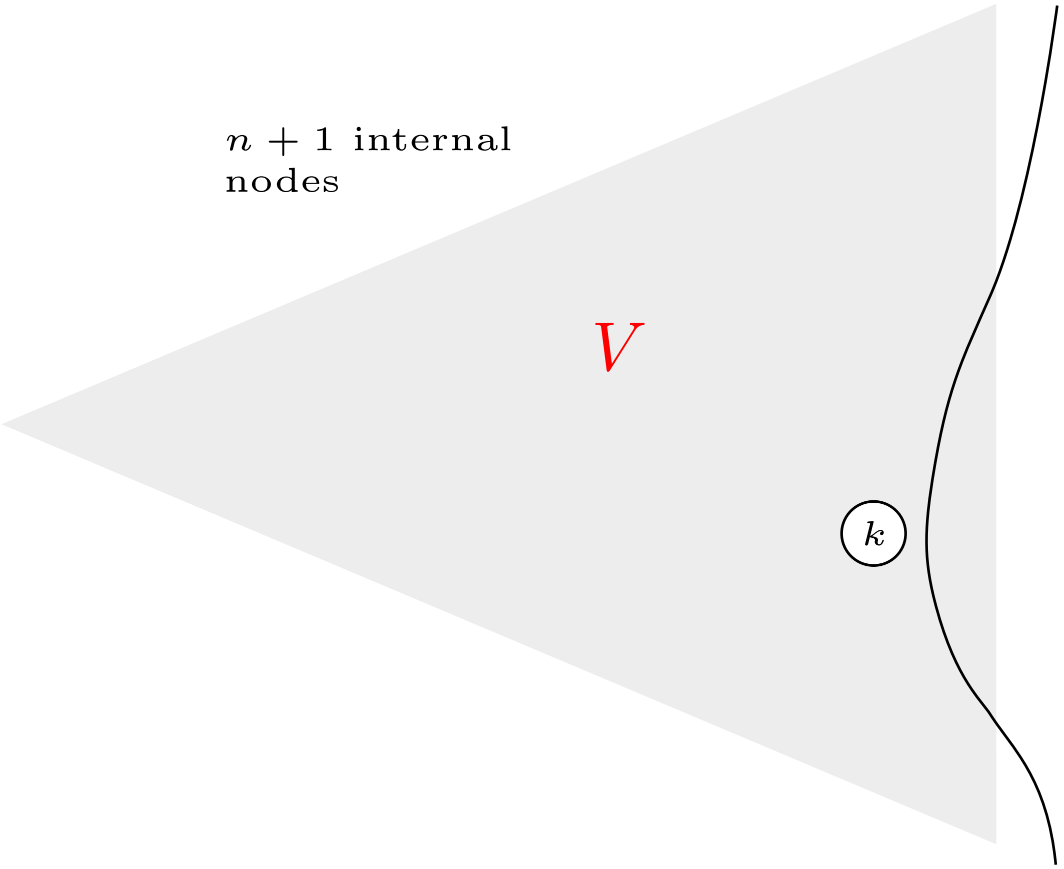
3. Assume the lemma is true for all code trees with n internal nodes, and consider a code tree with $n+1$ internal nodes.

4. Let k be an internal node such that k is the parent of a leaf c with maximum order.

5. Each sibling of c may or may not be a leaf. If it is not a leaf, then it cannot be the ascendent of another leaf because we assume that c is a leaf with maximum order.

6. Now consider revealing the outcome of X in two steps. In the first step, if the outcome of X is not a leaf descending from node k, we identify the outcome exactly, otherwise we identify the outcome to be a child of node k. We call this random variable V.

7. If we do not identify the outcome exactly in the first step, which happens with probability q_k, we further identify in the second step which of the children (child) of node k the outcome is (there is only one child of node k which can be the outcome if all the siblings of c are not leaves). We call this random variable W.



Lemma 4.19 $H_D(X) = \sum_{k \in \mathcal{I}} q_k h_k$.

Proof

1. We prove the lemma by induction on the number of internal nodes of the code tree.

2. If there is only one internal node, it must be the root of the tree. Then the lemma is trivially true upon observing that the reaching probability of the root is equal to 1.

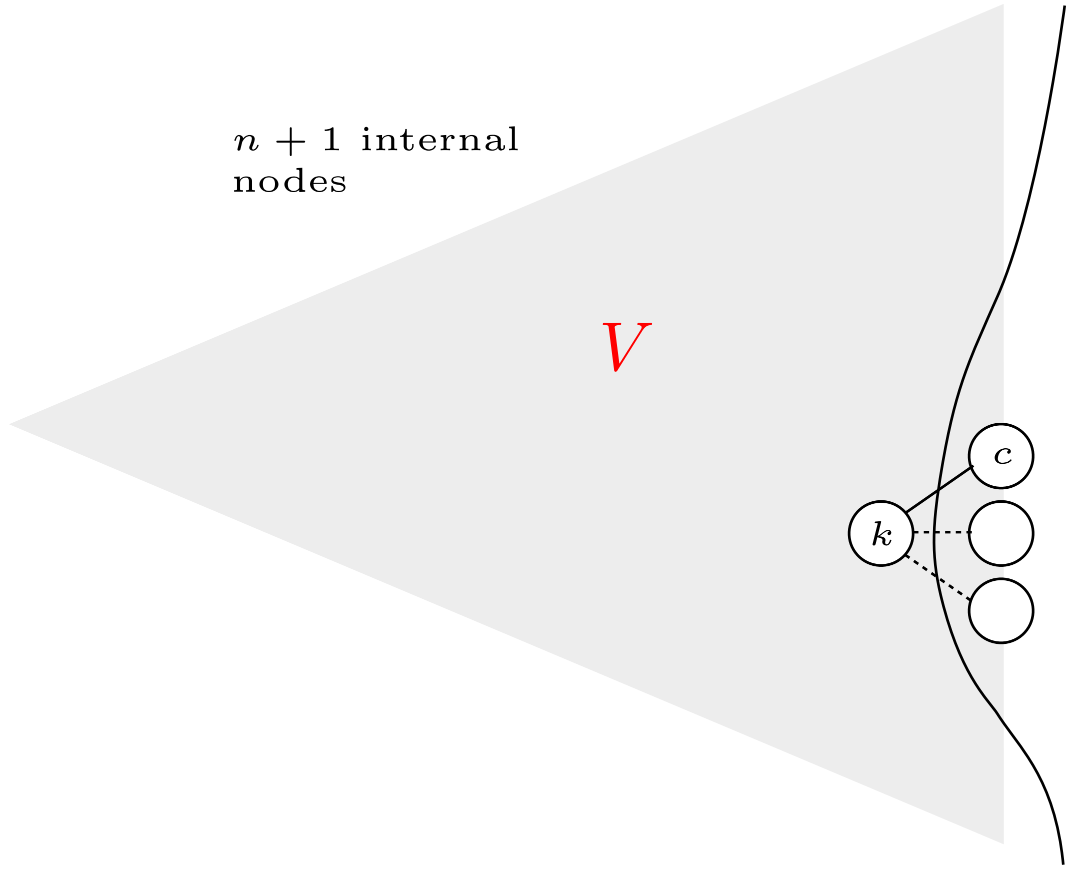
3. Assume the lemma is true for all code trees with n internal nodes, and consider a code tree with $n + 1$ internal nodes.

4. Let k be an internal node such that k is the parent of a leaf c with maximum order.

5. Each sibling of c may or may not be a leaf. If it is not a leaf, then it cannot be the ascendent of another leaf because we assume that c is a leaf with maximum order.

6. Now consider revealing the outcome of X in two steps. In the first step, if the outcome of X is not a leaf descending from node k, we identify the outcome exactly, otherwise we identify the outcome to be a child of node k. We call this random variable V.

7. If we do not identify the outcome exactly in the first step, which happens with probability q_k, we further identify in the second step which of the children (child) of node k the outcome is (there is only one child of node k which can be the outcome if all the siblings of c are not leaves). We call this random variable W.



Lemma 4.19 $H_D(X) = \sum_{k \in \mathcal{I}} q_k h_k$.

Proof

1. We prove the lemma by induction on the number of internal nodes of the code tree.

2. If there is only one internal node, it must be the root of the tree. Then the lemma is trivially true upon observing that the reaching probability of the root is equal to 1.

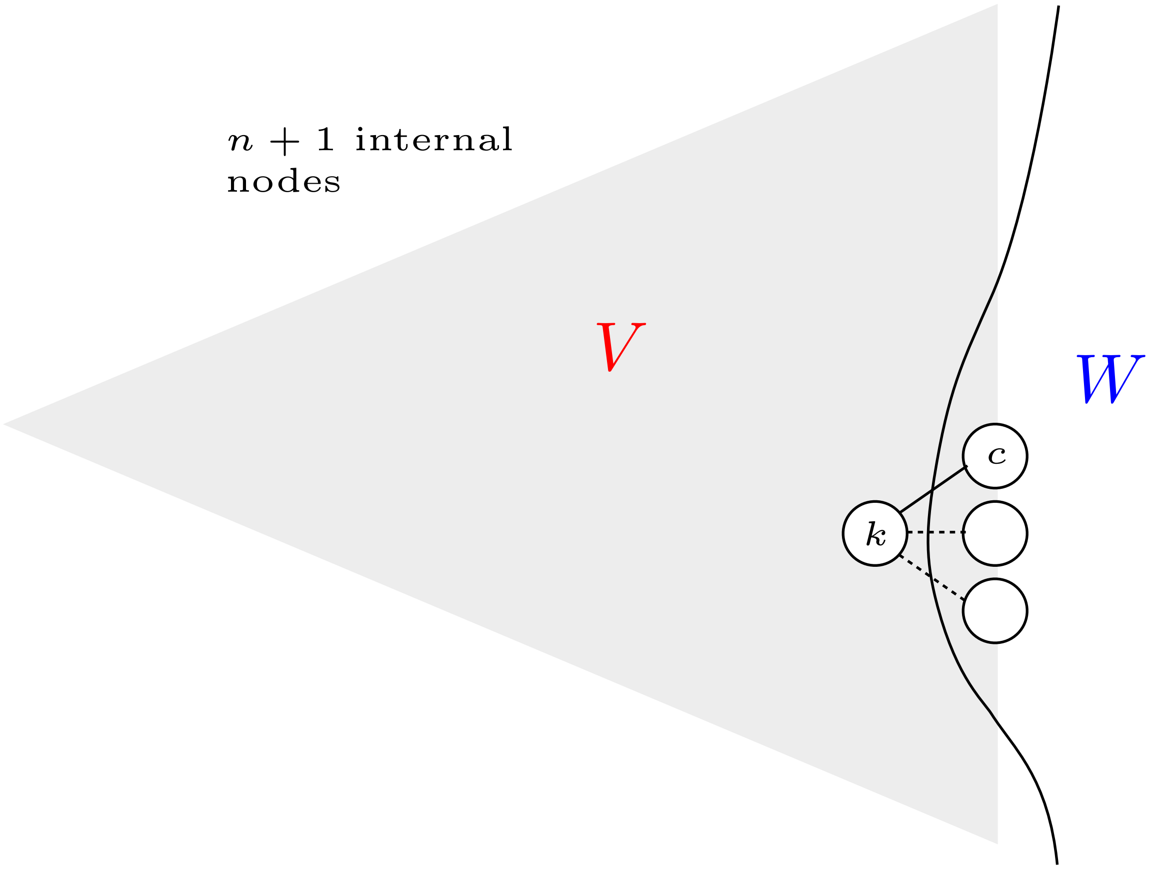
3. Assume the lemma is true for all code trees with n internal nodes, and consider a code tree with $n + 1$ internal nodes.

4. Let k be an internal node such that k is the parent of a leaf c with maximum order.

5. Each sibling of c may or may not be a leaf. If it is not a leaf, then it cannot be the ascendent of another leaf because we assume that c is a leaf with maximum order.

6. Now consider revealing the outcome of X in two steps. In the first step, if the outcome of X is not a leaf descending from node k, we identify the outcome exactly, otherwise we identify the outcome to be a child of node k. We call this random variable V.

7. If we do not identify the outcome exactly in the first step, which happens with probability q_k, we further identify in the second step which of the children (child) of node k the outcome is (there is only one child of node k which can be the outcome if all the siblings of c are not leaves). We call this random variable W.



Lemma 4.19 $H_D(X) = \sum_{k \in \mathcal{I}} q_k h_k$.

Proof

1. We prove the lemma by induction on the number of internal nodes of the code tree.

2. If there is only one internal node, it must be the root of the tree. Then the lemma is trivially true upon observing that the reaching probability of the root is equal to 1.

3. Assume the lemma is true for all code trees with n internal nodes, and consider a code tree with $n + 1$ internal nodes.

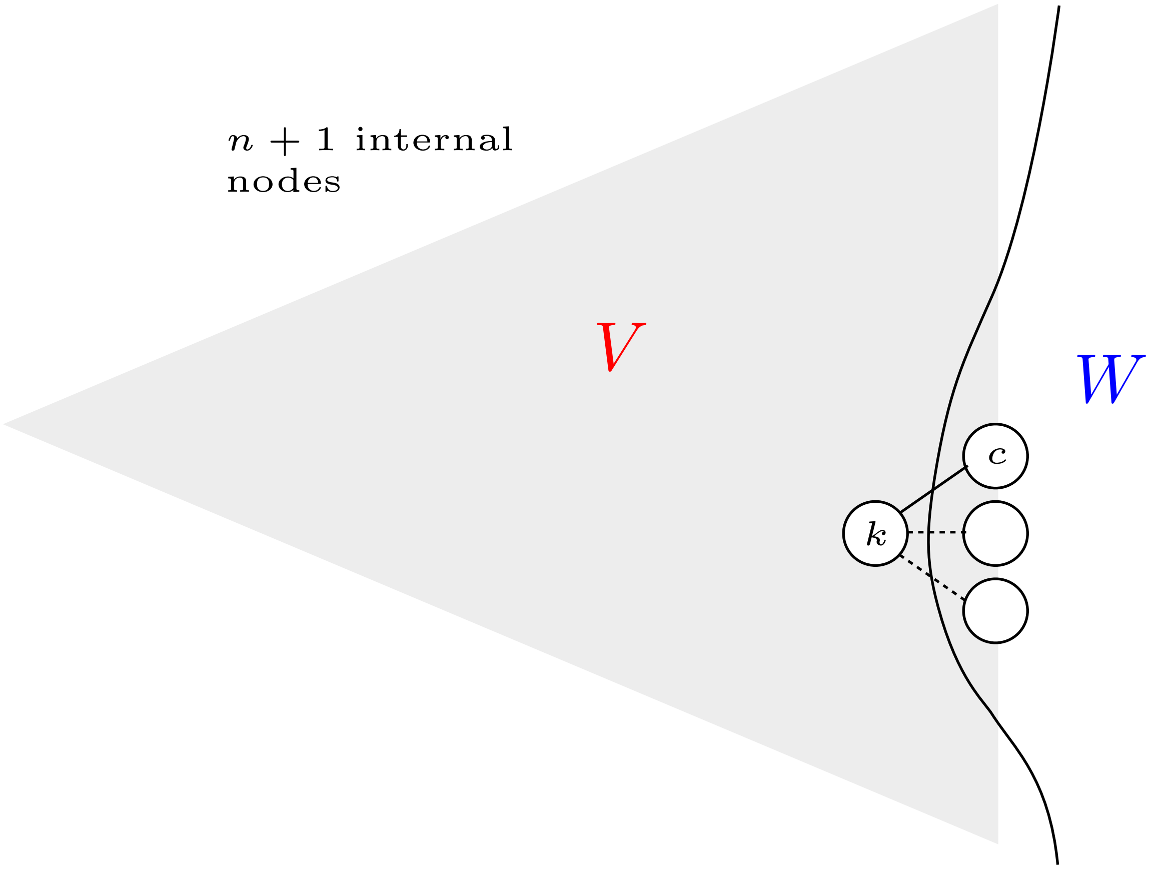
4. Let k be an internal node such that k is the parent of a leaf c with maximum order.

5. Each sibling of c may or may not be a leaf. If it is not a leaf, then it cannot be the ascendent of another leaf because we assume that c is a leaf with maximum order.

6. Now consider revealing the outcome of X in two steps. In the first step, if the outcome of X is not a leaf descending from node k, we identify the outcome exactly, otherwise we identify the outcome to be a child of node k. We call this random variable V.

7. If we do not identify the outcome exactly in the first step, which happens with probability q_k, we further identify in the second step which of the children (child) of node k the outcome is (there is only one child of node k which can be the outcome if all the siblings of c are not leaves). We call this random variable W.

8. If the second step is not necessary, we assume that W takes a constant value with probability 1.



Lemma 4.19 $H_D(X) = \sum_{k \in \mathcal{I}} q_k h_k$.

Proof

1. We prove the lemma by induction on the number of internal nodes of the code tree.

2. If there is only one internal node, it must be the root of the tree. Then the lemma is trivially true upon observing that the reaching probability of the root is equal to 1.

3. Assume the lemma is true for all code trees with n internal nodes, and consider a code tree with $n + 1$ internal nodes.

4. Let k be an internal node such that k is the parent of a leaf c with maximum order.

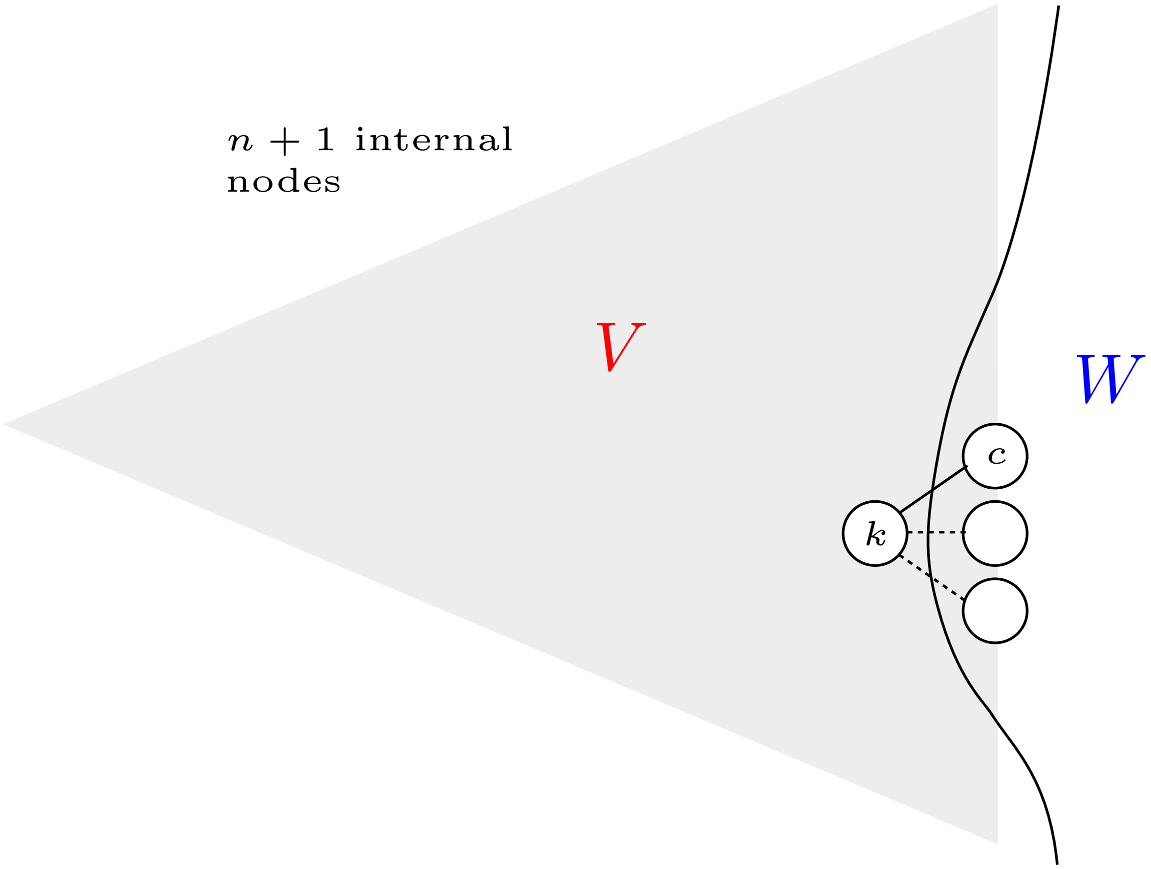
5. Each sibling of c may or may not be a leaf. If it is not a leaf, then it cannot be the ascendent of another leaf because we assume that c is a leaf with maximum order.

6. Now consider revealing the outcome of X in two steps. In the first step, if the outcome of X is not a leaf descending from node k, we identify the outcome exactly, otherwise we identify the outcome to be a child of node k. We call this random variable V.

7. If we do not identify the outcome exactly in the first step, which happens with probability q_k, we further identify in the second step which of the children (child) of node k the outcome is (there is only one child of node k which can be the outcome if all the siblings of c are not leaves). We call this random variable W.

8. If the second step is not necessary, we assume that W takes a constant value with probability 1.

9. Then $X = (V, W)$.



Lemma 4.19 $H_D(X) = \sum_{k \in \mathcal{I}} q_k h_k$.

Proof

1. We prove the lemma by induction on the number of internal nodes of the code tree.

2. If there is only one internal node, it must be the root of the tree. Then the lemma is trivially true upon observing that the reaching probability of the root is equal to 1.

3. Assume the lemma is true for all code trees with n internal nodes, and consider a code tree with $n + 1$ internal nodes.

4. Let k be an internal node such that k is the parent of a leaf c with maximum order.

5. Each sibling of c may or may not be a leaf. If it is not a leaf, then it cannot be the ascendent of another leaf because we assume that c is a leaf with maximum order.

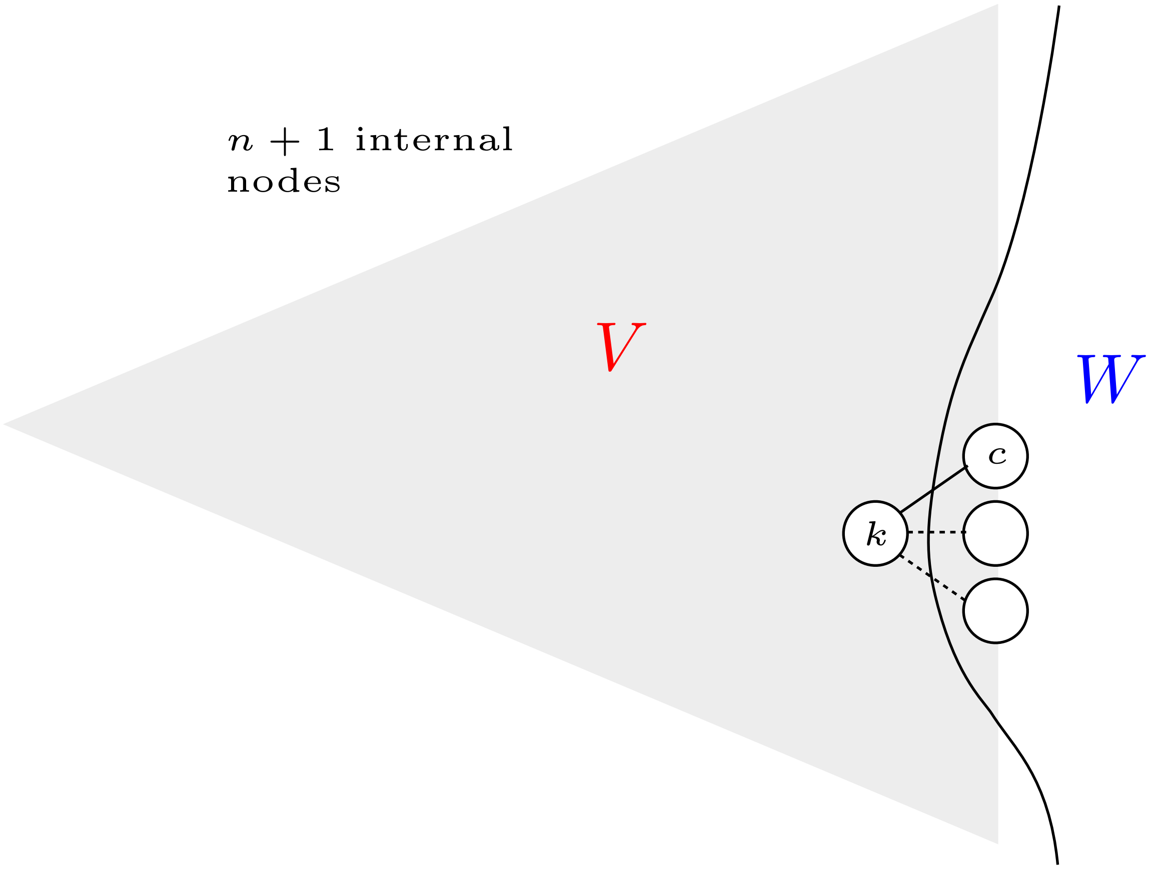
6. Now consider revealing the outcome of X in two steps. In the first step, if the outcome of X is not a leaf descending from node k, we identify the outcome exactly, otherwise we identify the outcome to be a child of node k. We call this random variable V.

7. If we do not identify the outcome exactly in the first step, which happens with probability q_k, we further identify in the second step which of the children (child) of node k the outcome is (there is only one child of node k which can be the outcome if all the siblings of c are not leaves). We call this random variable W.

8. If the second step is not necessary, we assume that W takes a constant value with probability 1.

9. Then $X = (V, W)$.

10. The outcome of V can be represented by a code tree with n internal nodes which is obtained by pruning the original code tree at node k.



Lemma 4.19 $H_D(X) = \sum_{k \in \mathcal{I}} q_k h_k$.

Proof

1. We prove the lemma by induction on the number of internal nodes of the code tree.

2. If there is only one internal node, it must be the root of the tree. Then the lemma is trivially true upon observing that the reaching probability of the root is equal to 1.

3. Assume the lemma is true for all code trees with n internal nodes, and consider a code tree with $n+1$ internal nodes.

4. Let k be an internal node such that k is the parent of a leaf c with maximum order.

5. Each sibling of c may or may not be a leaf. If it is not a leaf, then it cannot be the ascendent of another leaf because we assume that c is a leaf with maximum order.

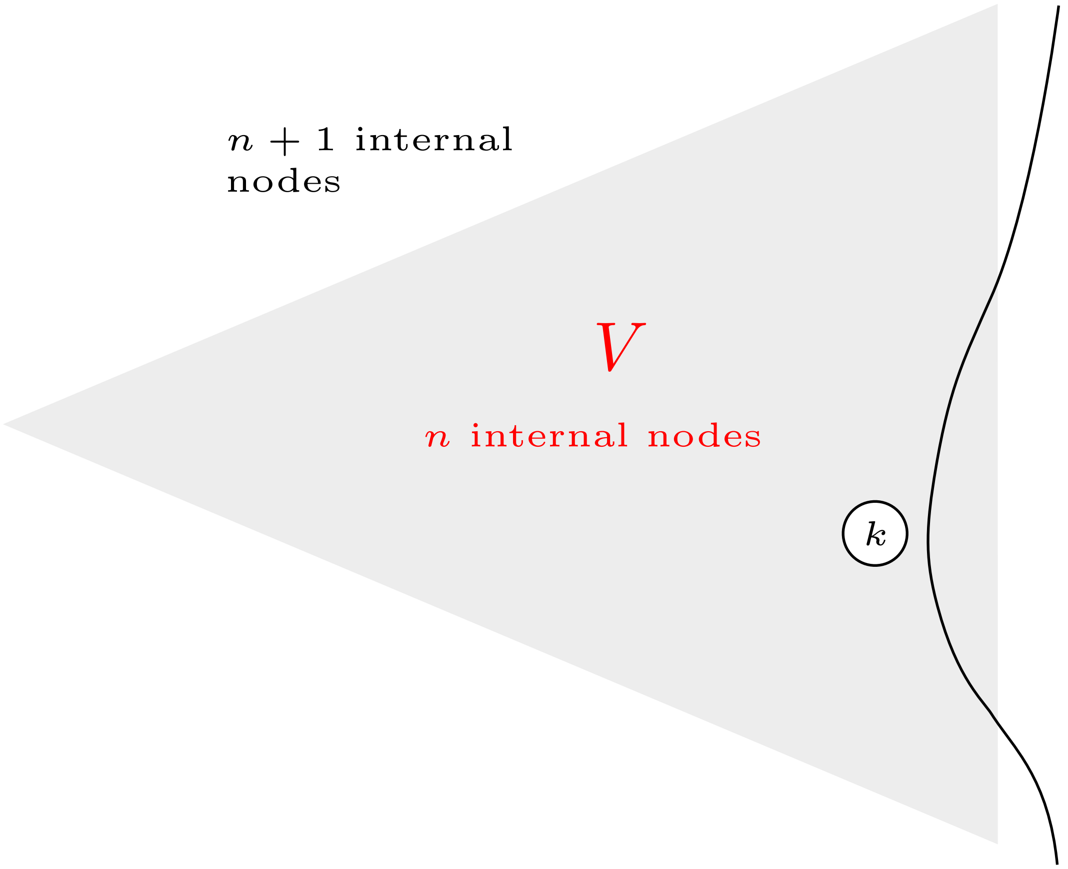
6. Now consider revealing the outcome of X in two steps. In the first step, if the outcome of X is not a leaf descending from node k, we identify the outcome exactly, otherwise we identify the outcome to be a child of node k. We call this random variable V.

7. If we do not identify the outcome exactly in the first step, which happens with probability q_k, we further identify in the second step which of the children (child) of node k the outcome is (there is only one child of node k which can be the outcome if all the siblings of c are not leaves). We call this random variable W.

8. If the second step is not necessary, we assume that W takes a constant value with probability 1.

9. Then $X = (V, W)$.

10. The outcome of V can be represented by a code tree with n internal nodes which is obtained by pruning the original code tree at node k.



Lemma 4.19 $H_D(X) = \sum_{k \in \mathcal{I}} q_k h_k$.

Proof

1. We prove the lemma by induction on the number of internal nodes of the code tree.

2. If there is only one internal node, it must be the root of the tree. Then the lemma is trivially true upon observing that the reaching probability of the root is equal to 1.

3. Assume the lemma is true for all code trees with n internal nodes, and consider a code tree with $n + 1$ internal nodes.

4. Let k be an internal node such that k is the parent of a leaf c with maximum order.

5. Each sibling of c may or may not be a leaf. If it is not a leaf, then it cannot be the ascendent of another leaf because we assume that c is a leaf with maximum order.

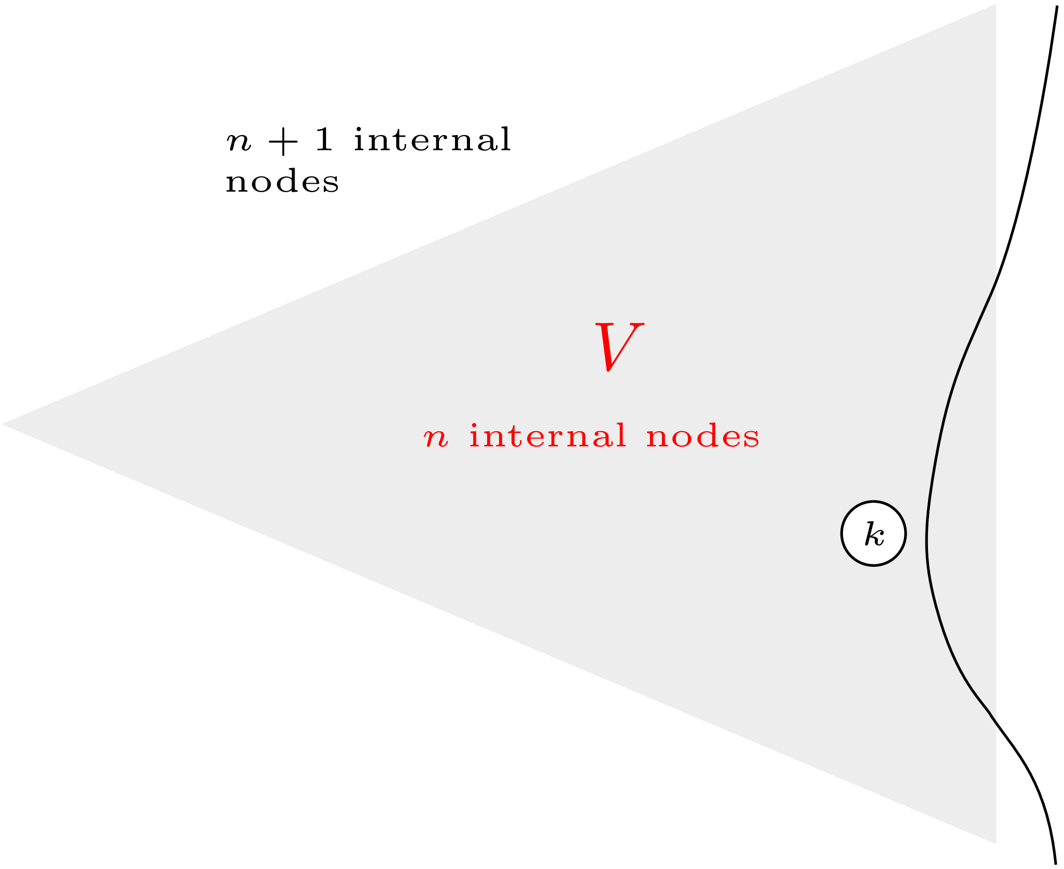
6. Now consider revealing the outcome of X in two steps. In the first step, if the outcome of X is not a leaf descending from node k, we identify the outcome exactly, otherwise we identify the outcome to be a child of node k. We call this random variable V.

7. If we do not identify the outcome exactly in the first step, which happens with probability q_k, we further identify in the second step which of the children (child) of node k the outcome is (there is only one child of node k which can be the outcome if all the siblings of c are not leaves). We call this random variable W.

8. If the second step is not necessary, we assume that W takes a constant value with probability 1.

9. Then $X = (V, W)$.

10. The outcome of V can be represented by a code tree with n internal nodes which is obtained by pruning the original code tree at node k.



Lemma 4.19 $H_D(X) = \sum_{k \in \mathcal{I}} q_k h_k$.

Proof

1. We prove the lemma by induction on the number of internal nodes of the code tree.

2. If there is only one internal node, it must be the root of the tree. Then the lemma is trivially true upon observing that the reaching probability of the root is equal to 1.

3. Assume the lemma is true for all code trees with n internal nodes, and consider a code tree with $n+1$ internal nodes.

4. Let k be an internal node such that k is the parent of a leaf c with maximum order.

5. Each sibling of c may or may not be a leaf. If it is not a leaf, then it cannot be the ascendent of another leaf because we assume that c is a leaf with maximum order.

6. Now consider revealing the outcome of X in two steps. In the first step, if the outcome of X is not a leaf descending from node k, we identify the outcome exactly, otherwise we identify the outcome to be a child of node k. We call this random variable V.

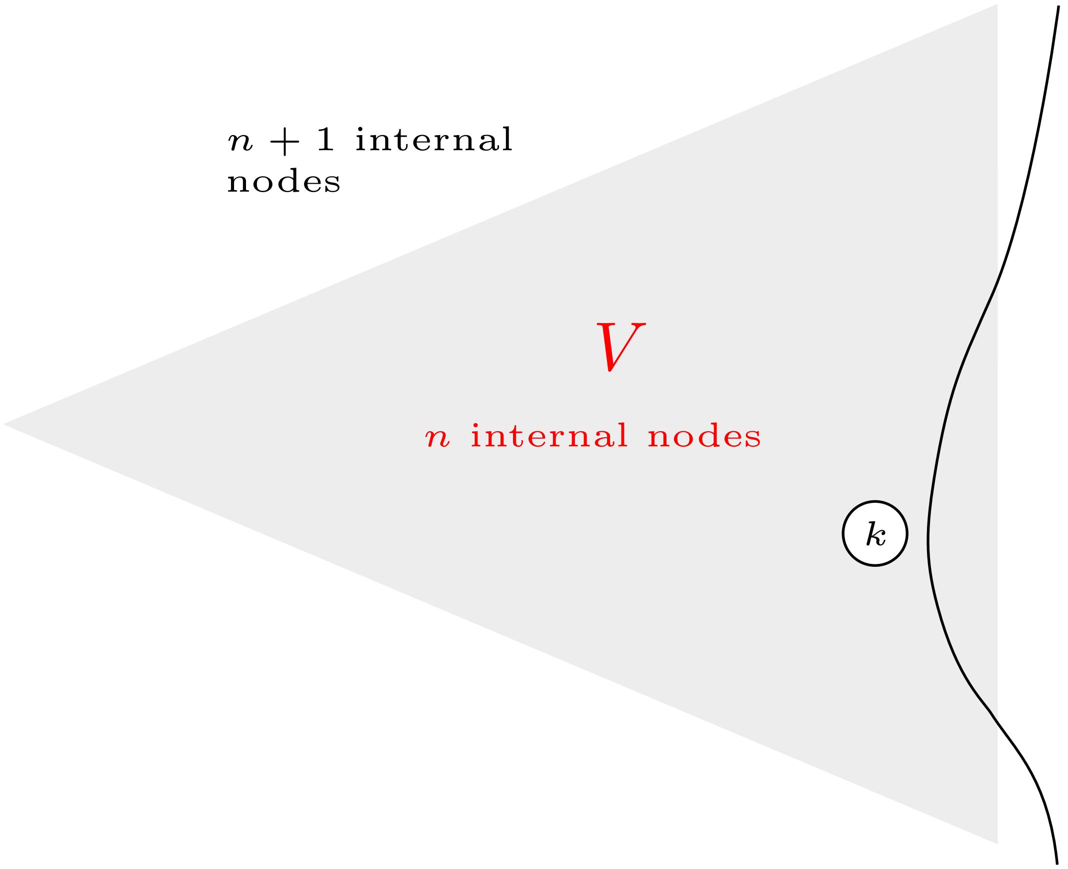
7. If we do not identify the outcome exactly in the first step, which happens with probability q_k, we further identify in the second step which of the children (child) of node k the outcome is (there is only one child of node k which can be the outcome if all the siblings of c are not leaves). We call this random variable W.

8. If the second step is not necessary, we assume that W takes a constant value with probability 1.

9. Then $X = (V, W)$.

10. The outcome of V can be represented by a code tree with n internal nodes which is obtained by pruning the original code tree at node k.

11. By the induction hypothesis,



Lemma 4.19 $H_D(X) = \sum_{k \in \mathcal{I}} q_k h_k$.

Proof

1. We prove the lemma by induction on the number of internal nodes of the code tree.

2. If there is only one internal node, it must be the root of the tree. Then the lemma is trivially true upon observing that the reaching probability of the root is equal to 1.

3. Assume the lemma is true for all code trees with n internal nodes, and consider a code tree with $n+1$ internal nodes.

4. Let k be an internal node such that k is the parent of a leaf c with maximum order.

5. Each sibling of c may or may not be a leaf. If it is not a leaf, then it cannot be the ascendent of another leaf because we assume that c is a leaf with maximum order.

6. Now consider revealing the outcome of X in two steps. In the first step, if the outcome of X is not a leaf descending from node k, we identify the outcome exactly, otherwise we identify the outcome to be a child of node k. We call this random variable V.

7. If we do not identify the outcome exactly in the first step, which happens with probability q_k, we further identify in the second step which of the children (child) of node k the outcome is (there is only one child of node k which can be the outcome if all the siblings of c are not leaves). We call this random variable W.

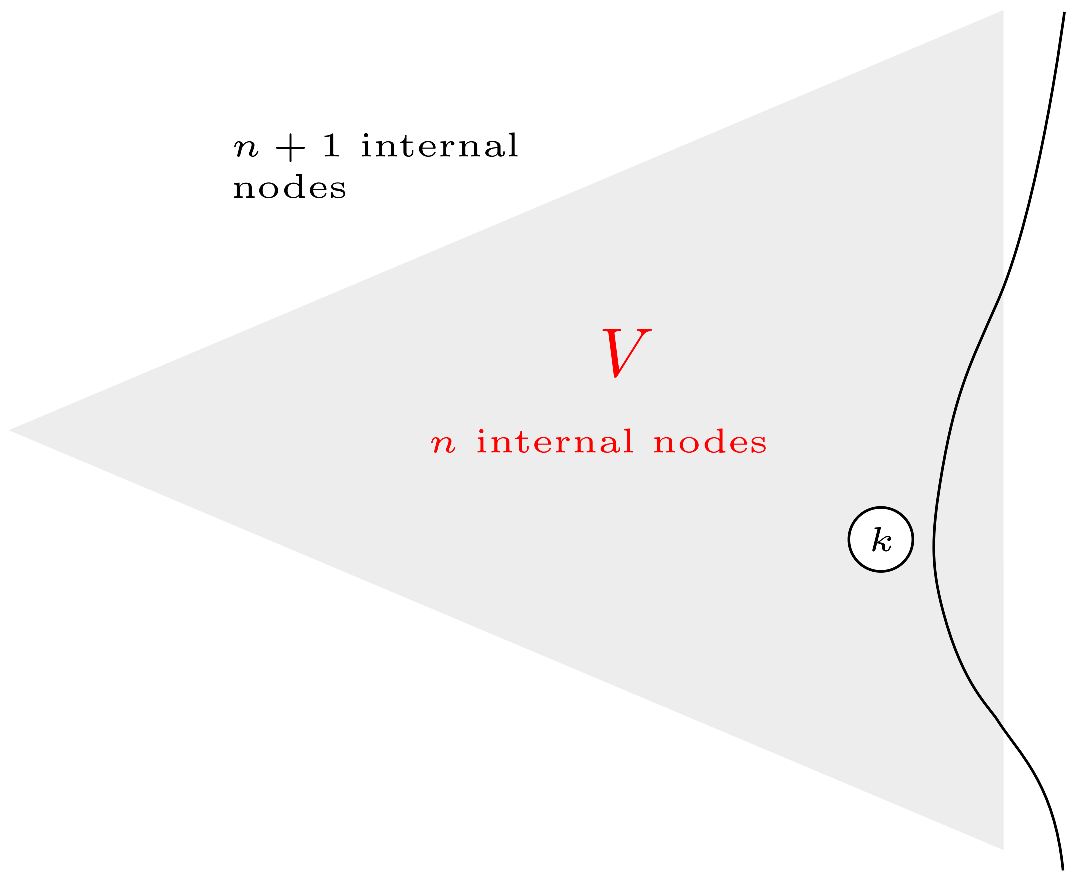
8. If the second step is not necessary, we assume that W takes a constant value with probability 1.

9. Then $X = (V, W)$.

10. The outcome of V can be represented by a code tree with n internal nodes which is obtained by pruning the original code tree at node k.

11. By the induction hypothesis,

$$H(V) = \sum_{k' \in \mathcal{I} \setminus \{k\}} q_{k'} h_{k'}.$$



Lemma 4.19 $H_D(X) = \sum_{k \in \mathcal{I}} q_k h_k$.

Proof

1. We prove the lemma by induction on the number of internal nodes of the code tree.

2. If there is only one internal node, it must be the root of the tree. Then the lemma is trivially true upon observing that the reaching probability of the root is equal to 1.

3. Assume the lemma is true for all code trees with n internal nodes, and consider a code tree with $n + 1$ internal nodes.

4. Let k be an internal node such that k is the parent of a leaf c with maximum order.

5. Each sibling of c may or may not be a leaf. If it is not a leaf, then it cannot be the ascendent of another leaf because we assume that c is a leaf with maximum order.

6. Now consider revealing the outcome of X in two steps. In the first step, if the outcome of X is not a leaf descending from node k, we identify the outcome exactly, otherwise we identify the outcome to be a child of node k. We call this random variable V.

7. If we do not identify the outcome exactly in the first step, which happens with probability q_k, we further identify in the second step which of the children (child) of node k the outcome is (there is only one child of node k which can be the outcome if all the siblings of c are not leaves). We call this random variable W.

8. If the second step is not necessary, we assume that W takes a constant value with probability 1.

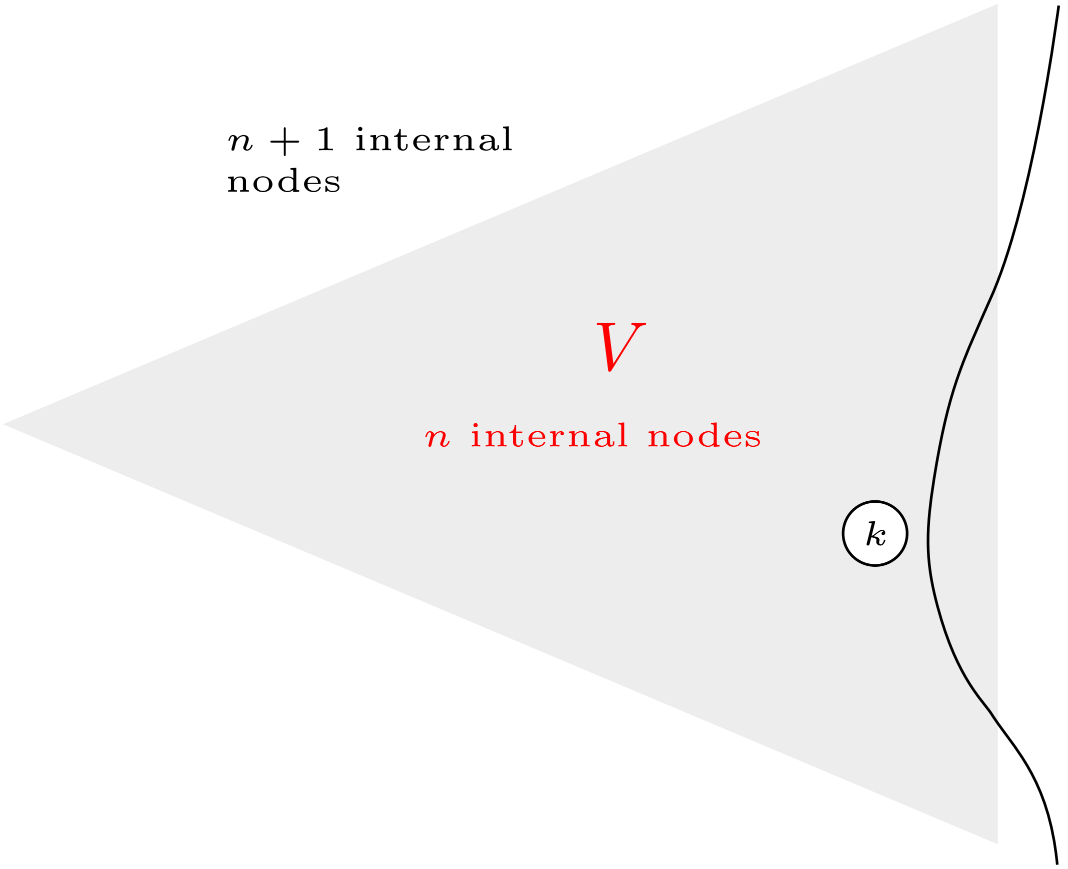
9. Then $X = (V, W)$.

10. The outcome of V can be represented by a code tree with n internal nodes which is obtained by pruning the original code tree at node k.

11. By the induction hypothesis,

$$H(V) = \sum_{k' \in \mathcal{I} \setminus \{k\}} q_{k'} h_{k'}.$$

12. Then we have



Lemma 4.19 $H_D(X) = \sum_{k \in \mathcal{I}} q_k h_k$.

Proof

1. We prove the lemma by induction on the number of internal nodes of the code tree.

2. If there is only one internal node, it must be the root of the tree. Then the lemma is trivially true upon observing that the reaching probability of the root is equal to 1.

3. Assume the lemma is true for all code trees with n internal nodes, and consider a code tree with $n+1$ internal nodes.

4. Let k be an internal node such that k is the parent of a leaf c with maximum order.

5. Each sibling of c may or may not be a leaf. If it is not a leaf, then it cannot be the ascendent of another leaf because we assume that c is a leaf with maximum order.

6. Now consider revealing the outcome of X in two steps. In the first step, if the outcome of X is not a leaf descending from node k, we identify the outcome exactly, otherwise we identify the outcome to be a child of node k. We call this random variable V.

7. If we do not identify the outcome exactly in the first step, which happens with probability q_k, we further identify in the second step which of the children (child) of node k the outcome is (there is only one child of node k which can be the outcome if all the siblings of c are not leaves). We call this random variable W.

8. If the second step is not necessary, we assume that W takes a constant value with probability 1.

9. Then $X = (V, W)$.

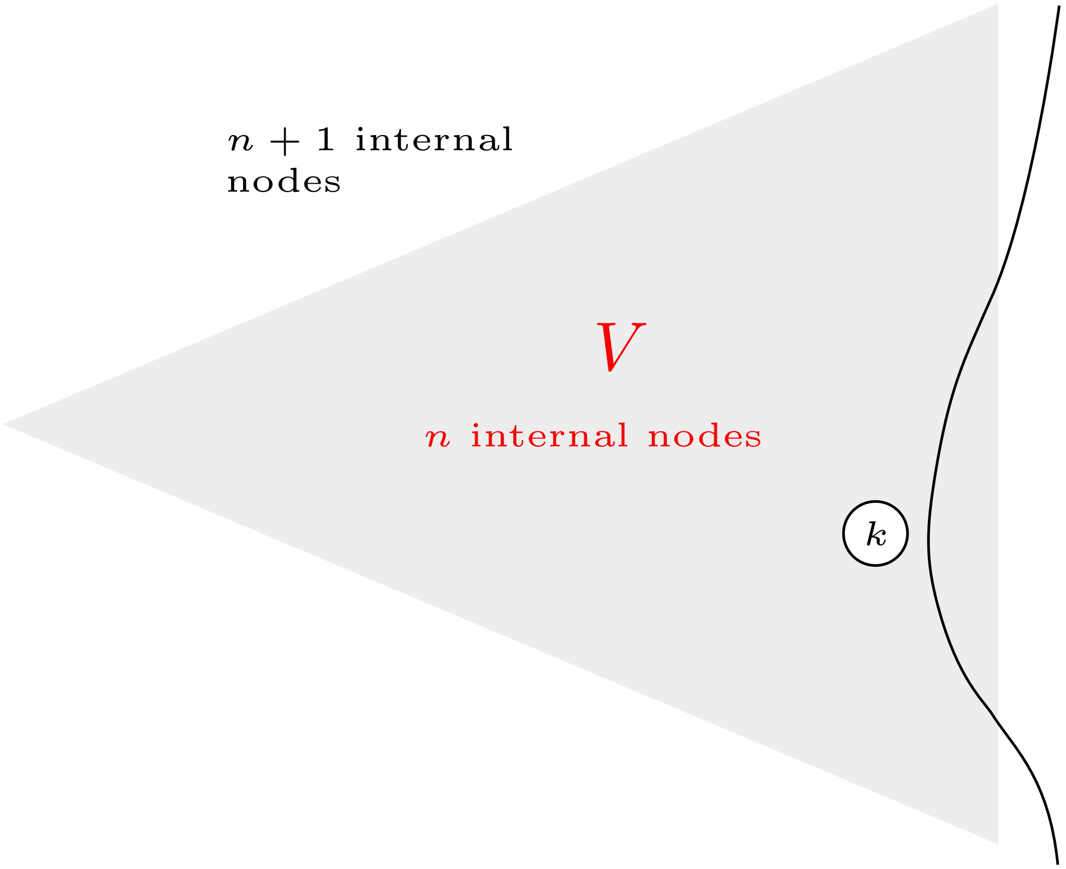
10. The outcome of V can be represented by a code tree with n internal nodes which is obtained by pruning the original code tree at node k.

11. By the induction hypothesis,

$$H(V) = \sum_{k' \in \mathcal{I} \setminus \{k\}} q_{k'} h_{k'}.$$

12. Then we have

$$H(X) \quad = \quad H(V, W)$$



Lemma 4.19 $H_D(X) = \sum_{k \in \mathcal{I}} q_k h_k$.

Proof

1. We prove the lemma by induction on the number of internal nodes of the code tree.

2. If there is only one internal node, it must be the root of the tree. Then the lemma is trivially true upon observing that the reaching probability of the root is equal to 1.

3. Assume the lemma is true for all code trees with n internal nodes, and consider a code tree with $n+1$ internal nodes.

4. Let k be an internal node such that k is the parent of a leaf c with maximum order.

5. Each sibling of c may or may not be a leaf. If it is not a leaf, then it cannot be the ascendent of another leaf because we assume that c is a leaf with maximum order.

6. Now consider revealing the outcome of X in two steps. In the first step, if the outcome of X is not a leaf descending from node k, we identify the outcome exactly, otherwise we identify the outcome to be a child of node k. We call this random variable V.

7. If we do not identify the outcome exactly in the first step, which happens with probability q_k, we further identify in the second step which of the children (child) of node k the outcome is (there is only one child of node k which can be the outcome if all the siblings of c are not leaves). We call this random variable W.

8. If the second step is not necessary, we assume that W takes a constant value with probability 1.

9. Then $X = (V, W)$.

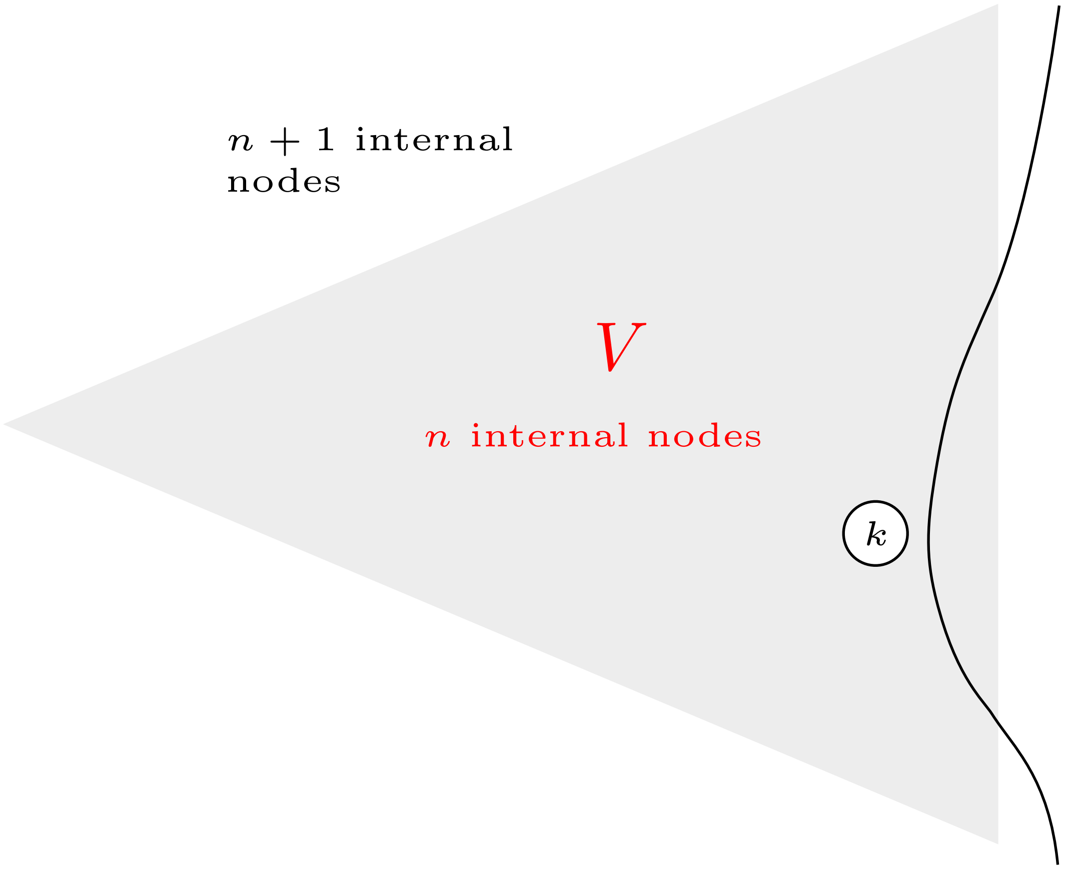
10. The outcome of V can be represented by a code tree with n internal nodes which is obtained by pruning the original code tree at node k.

11. By the induction hypothesis,

$$H(V) = \sum_{k' \in \mathcal{I} \backslash \{k\}} q_{k'} h_{k'}.$$

12. Then we have

$$\begin{aligned} H(X) &= H(V, W) \\ &= H(V) + H(W|V) \end{aligned}$$



Lemma 4.19 $H_D(X) = \sum_{k\in\mathcal{I}} q_k h_k$.

Proof

1. We prove the lemma by induction on the number of internal nodes of the code tree.

2. If there is only one internal node, it must be the root of the tree. Then the lemma is trivially true upon observing that the reaching probability of the root is equal to 1.

3. Assume the lemma is true for all code trees with n internal nodes, and consider a code tree with $n+1$ internal nodes.

4. Let k be an internal node such that k is the parent of a leaf c with maximum order.

5. Each sibling of c may or may not be a leaf. If it is not a leaf, then it cannot be the ascendent of another leaf because we assume that c is a leaf with maximum order.

6. Now consider revealing the outcome of X in two steps. In the first step, if the outcome of X is not a leaf descending from node k, we identify the outcome exactly, otherwise we identify the outcome to be a child of node k. We call this random variable V.

7. If we do not identify the outcome exactly in the first step, which happens with probability q_k, we further identify in the second step which of the children (child) of node k the outcome is (there is only one child of node k which can be the outcome if all the siblings of c are not leaves). We call this random variable W.

8. If the second step is not necessary, we assume that W takes a constant value with probability 1.

9. Then $X = (V, W)$.

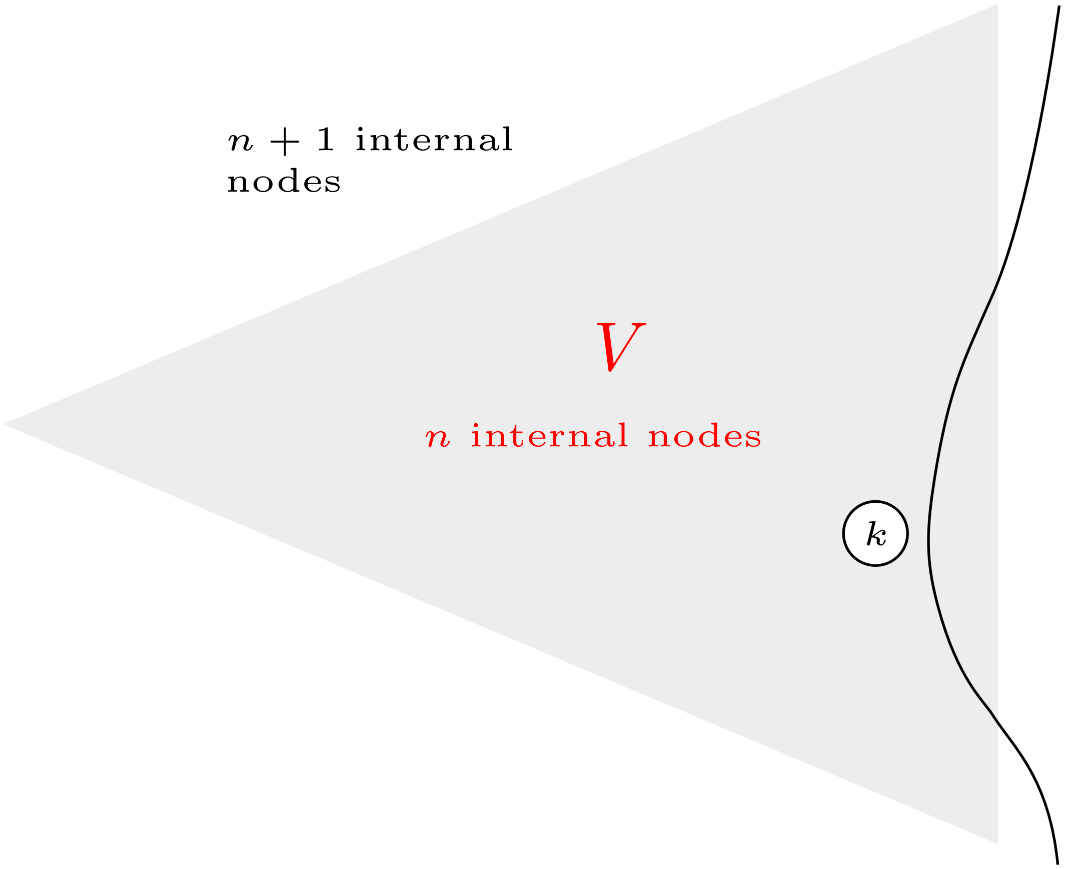
10. The outcome of V can be represented by a code tree with n internal nodes which is obtained by pruning the original code tree at node k.

11. By the induction hypothesis,

$$H(V) = \sum_{k'\in\mathcal{I}\backslash\{k\}} q_{k'} h_{k'}.$$

12. Then we have

$$\begin{aligned} H(X) &= H(V, W) \\ &= \underline{H(V)} + H(W|V) \end{aligned}$$



Lemma 4.19 $H_D(X) = \sum_{k\in\mathcal{I}} q_k h_k$.

Proof

1. We prove the lemma by induction on the number of internal nodes of the code tree.

2. If there is only one internal node, it must be the root of the tree. Then the lemma is trivially true upon observing that the reaching probability of the root is equal to 1.

3. Assume the lemma is true for all code trees with n internal nodes, and consider a code tree with $n+1$ internal nodes.

4. Let k be an internal node such that k is the parent of a leaf c with maximum order.

5. Each sibling of c may or may not be a leaf. If it is not a leaf, then it cannot be the ascendent of another leaf because we assume that c is a leaf with maximum order.

6. Now consider revealing the outcome of X in two steps. In the first step, if the outcome of X is not a leaf descending from node k, we identify the outcome exactly, otherwise we identify the outcome to be a child of node k. We call this random variable V.

7. If we do not identify the outcome exactly in the first step, which happens with probability q_k, we further identify in the second step which of the children (child) of node k the outcome is (there is only one child of node k which can be the outcome if all the siblings of c are not leaves). We call this random variable W.

8. If the second step is not necessary, we assume that W takes a constant value with probability 1.

9. Then $X = (V, W)$.

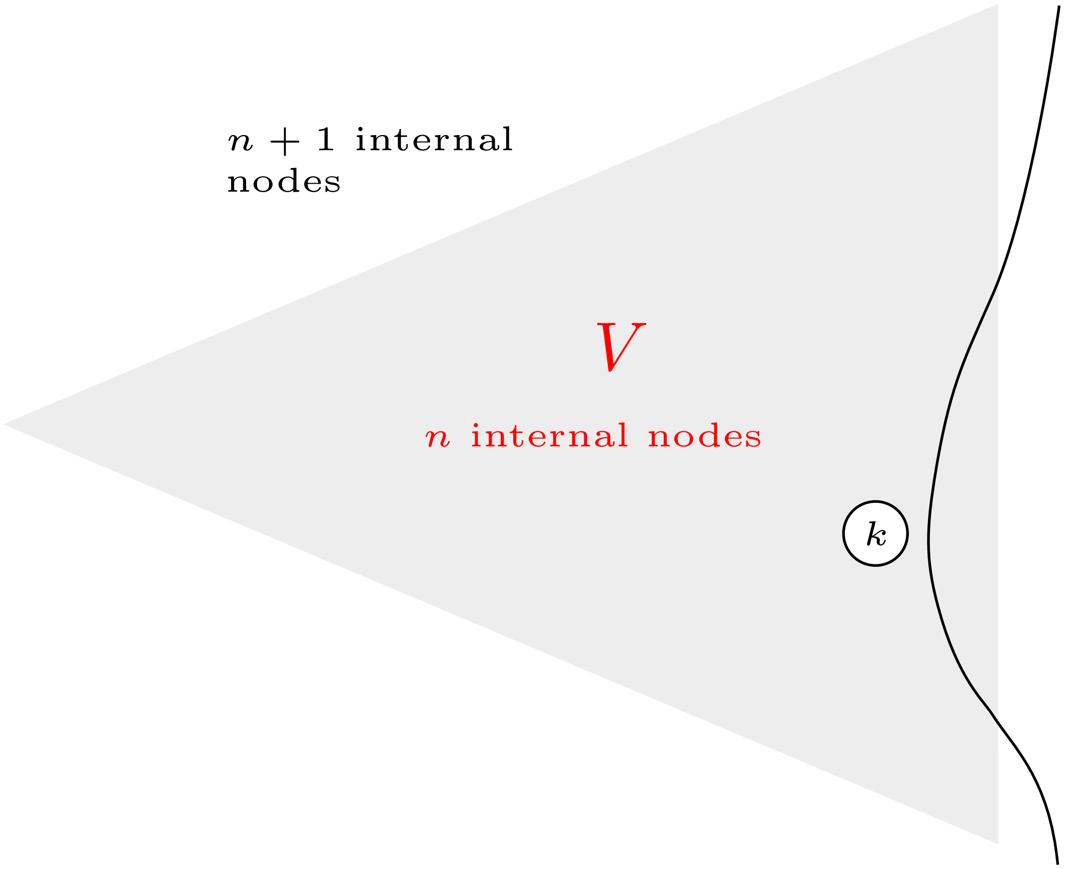
10. The outcome of V can be represented by a code tree with n internal nodes which is obtained by pruning the original code tree at node k.

11. By the induction hypothesis,

$$H(V) = \sum_{k'\in\mathcal{I}\backslash\{k\}} q_{k'} h_{k'}.$$

12. Then we have

$$\begin{aligned} H(X) &= H(V, W) \\ &= \underline{H(V)} + H(W|V) \\ &= \underline{\sum_{k'\in\mathcal{I}\backslash\{k\}} q_{k'} h_{k'}} + (1-q_k)\cdot 0 + q_k h_k \end{aligned}$$



Lemma 4.19 $H_D(X) = \sum_{k \in \mathcal{I}} q_k h_k$.

Proof

1. We prove the lemma by induction on the number of internal nodes of the code tree.

2. If there is only one internal node, it must be the root of the tree. Then the lemma is trivially true upon observing that the reaching probability of the root is equal to 1.

3. Assume the lemma is true for all code trees with n internal nodes, and consider a code tree with $n+1$ internal nodes.

4. Let k be an internal node such that k is the parent of a leaf c with maximum order.

5. Each sibling of c may or may not be a leaf. If it is not a leaf, then it cannot be the ascendent of another leaf because we assume that c is a leaf with maximum order.

6. Now consider revealing the outcome of X in two steps. In the first step, if the outcome of X is not a leaf descending from node k, we identify the outcome exactly, otherwise we identify the outcome to be a child of node k. We call this random variable V.

7. If we do not identify the outcome exactly in the first step, which happens with probability q_k, we further identify in the second step which of the children (child) of node k the outcome is (there is only one child of node k which can be the outcome if all the siblings of c are not leaves). We call this random variable W.

8. If the second step is not necessary, we assume that W takes a constant value with probability 1.

9. Then $X = (V, W)$.

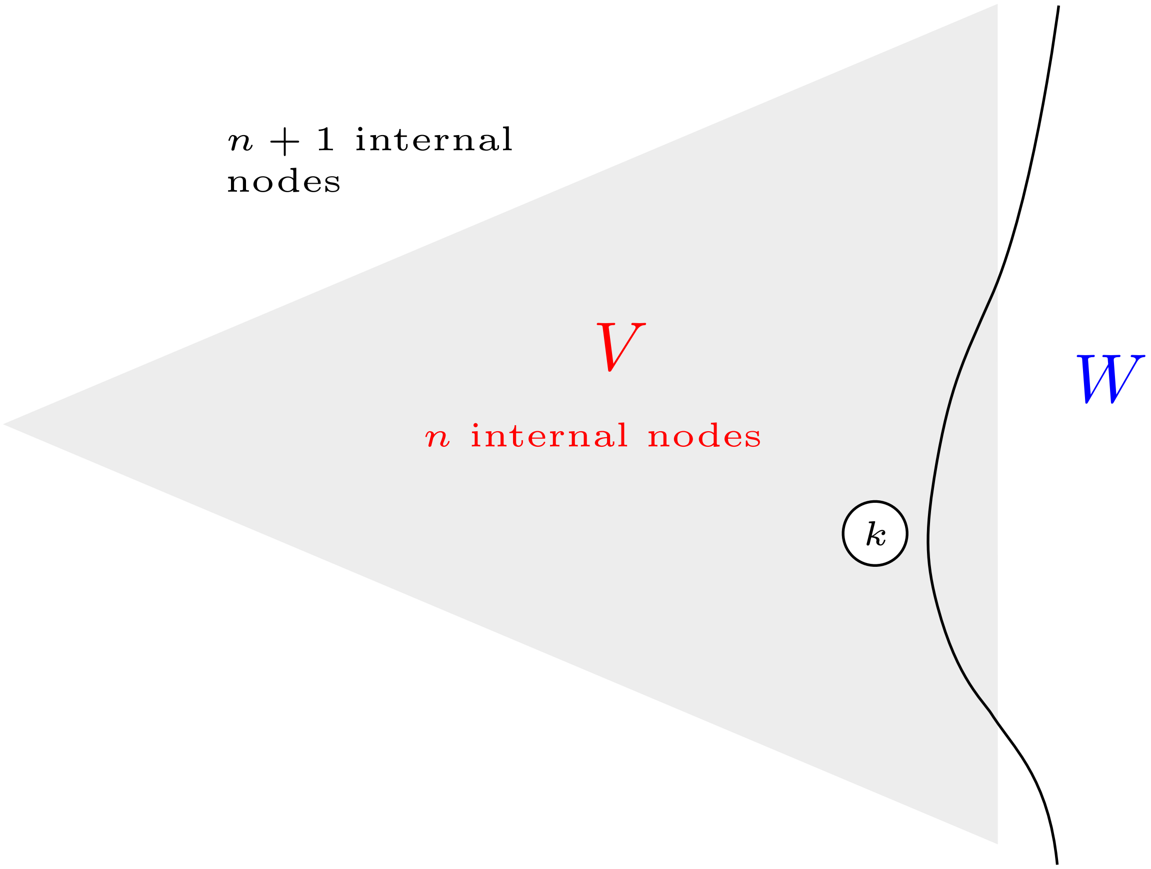
10. The outcome of V can be represented by a code tree with n internal nodes which is obtained by pruning the original code tree at node k.

11. By the induction hypothesis,

$$H(V) = \sum_{k' \in \mathcal{I} \backslash \{k\}} q_{k'} h_{k'}.$$

12. Then we have

$$\begin{aligned} H(X) &= H(V, W) \\ &= H(V) + \underline{H(W|V)} \\ &= \sum_{k' \in \mathcal{I} \backslash \{k\}} q_{k'} h_{k'} + (1 - q_k) \cdot 0 + q_k h_k \end{aligned}$$



Lemma 4.19 $H_D(X) = \sum_{k \in \mathcal{I}} q_k h_k$.

Proof

1. We prove the lemma by induction on the number of internal nodes of the code tree.

2. If there is only one internal node, it must be the root of the tree. Then the lemma is trivially true upon observing that the reaching probability of the root is equal to 1.

3. Assume the lemma is true for all code trees with n internal nodes, and consider a code tree with $n+1$ internal nodes.

4. Let k be an internal node such that k is the parent of a leaf c with maximum order.

5. Each sibling of c may or may not be a leaf. If it is not a leaf, then it cannot be the ascendent of another leaf because we assume that c is a leaf with maximum order.

6. Now consider revealing the outcome of X in two steps. In the first step, if the outcome of X is not a leaf descending from node k, we identify the outcome exactly, otherwise we identify the outcome to be a child of node k. We call this random variable V.

7. If we do not identify the outcome exactly in the first step, which happens with probability q_k, we further identify in the second step which of the children (child) of node k the outcome is (there is only one child of node k which can be the outcome if all the siblings of c are not leaves). We call this random variable W.

8. If the second step is not necessary, we assume that W takes a constant value with probability 1.

9. Then $X = (V, W)$.

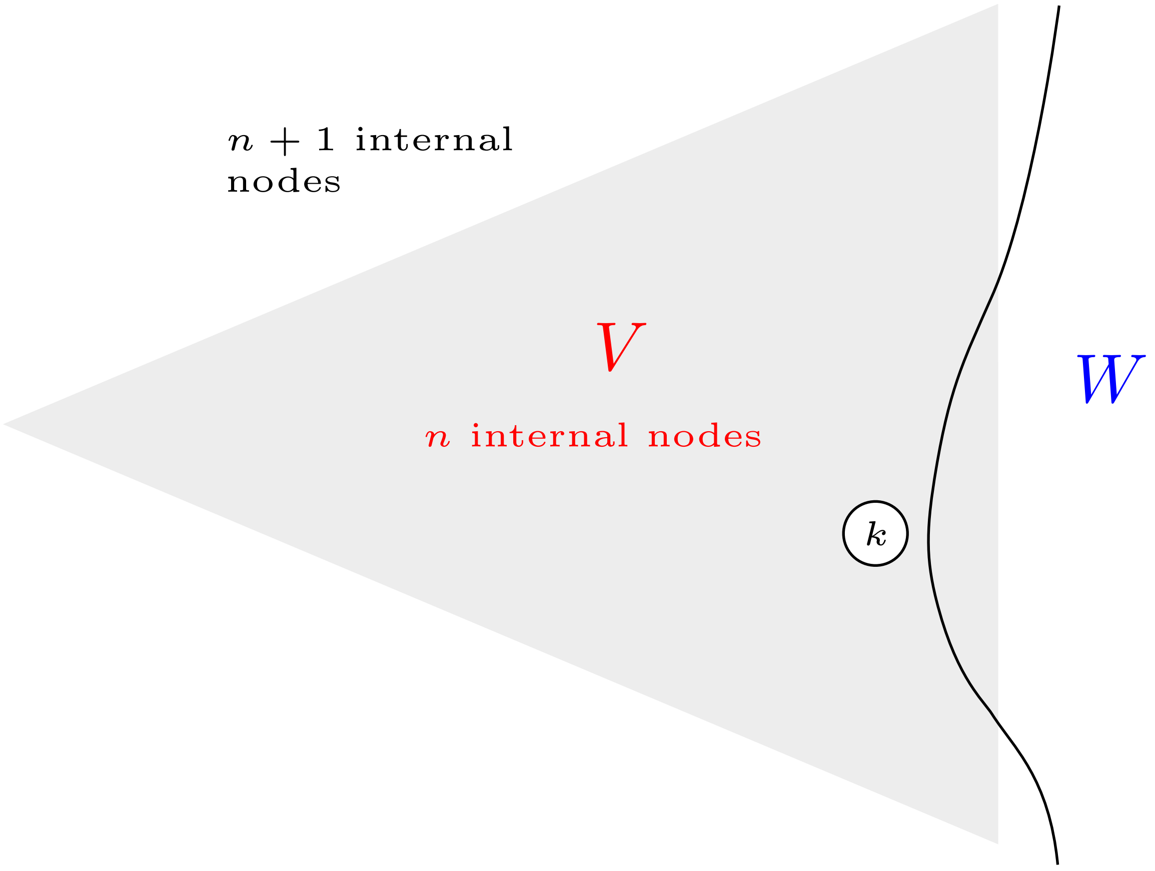
10. The outcome of V can be represented by a code tree with n internal nodes which is obtained by pruning the original code tree at node k.

11. By the induction hypothesis,

$$H(V) = \sum_{k' \in \mathcal{I} \backslash \{k\}} q_{k'} h_{k'}.$$

12. Then we have

$$\begin{aligned} H(X) &= H(V, W) \\ &= H(V) + \underline{H(W|V)} \\ &= \sum_{k' \in \mathcal{I} \backslash \{k\}} q_{k'} h_{k'} + \underline{(1 - q_k) \cdot 0} + q_k h_k \end{aligned}$$



Lemma 4.19 $H_D(X) = \sum_{k \in \mathcal{I}} q_k h_k$.

Proof

1. We prove the lemma by induction on the number of internal nodes of the code tree.

2. If there is only one internal node, it must be the root of the tree. Then the lemma is trivially true upon observing that the reaching probability of the root is equal to 1.

3. Assume the lemma is true for all code trees with n internal nodes, and consider a code tree with $n+1$ internal nodes.

4. Let k be an internal node such that k is the parent of a leaf c with maximum order.

5. Each sibling of c may or may not be a leaf. If it is not a leaf, then it cannot be the ascendent of another leaf because we assume that c is a leaf with maximum order.

6. Now consider revealing the outcome of X in two steps. In the first step, if the outcome of X is not a leaf descending from node k, we identify the outcome exactly, otherwise we identify the outcome to be a child of node k. We call this random variable V.

7. If we do not identify the outcome exactly in the first step, which happens with probability q_k, we further identify in the second step which of the children (child) of node k the outcome is (there is only one child of node k which can be the outcome if all the siblings of c are not leaves). We call this random variable W.

8. If the second step is not necessary, we assume that W takes a constant value with probability 1.

9. Then $X = (V, W)$.

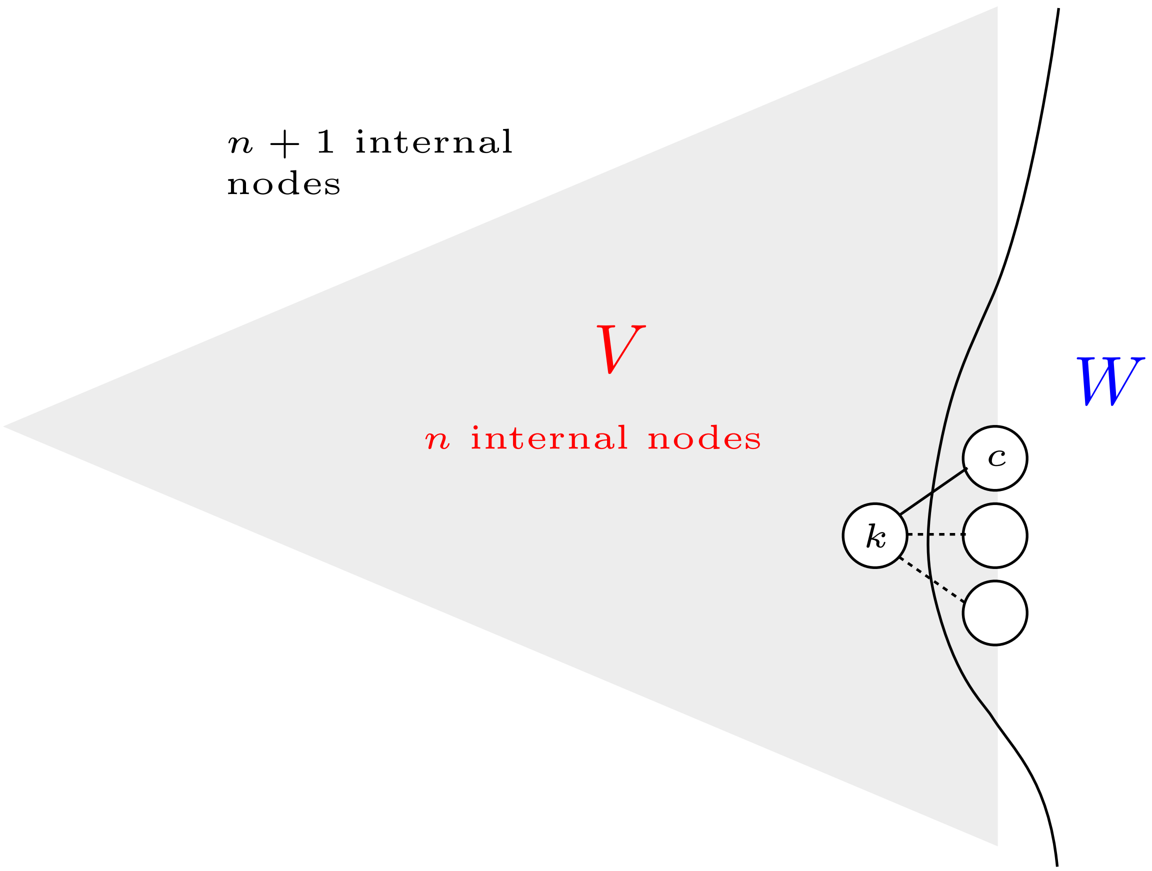
10. The outcome of V can be represented by a code tree with n internal nodes which is obtained by pruning the original code tree at node k.

11. By the induction hypothesis,

$$H(V) = \sum_{k' \in \mathcal{I} \backslash \{k\}} q_{k'} h_{k'}.$$

12. Then we have

$$\begin{aligned} H(X) &= H(V, W) \\ &= H(V) + \underline{H(W|V)} \\ &= \sum_{k' \in \mathcal{I} \backslash \{k\}} q_{k'} h_{k'} + (1 - q_k) \cdot 0 + \underline{q_k h_k} \end{aligned}$$



Lemma 4.19 $H_D(X) = \sum_{k\in\mathcal{I}} q_k h_k$.

Proof

1. We prove the lemma by induction on the number of internal nodes of the code tree.

2. If there is only one internal node, it must be the root of the tree. Then the lemma is trivially true upon observing that the reaching probability of the root is equal to 1.

3. Assume the lemma is true for all code trees with n internal nodes, and consider a code tree with $n+1$ internal nodes.

4. Let k be an internal node such that k is the parent of a leaf c with maximum order.

5. Each sibling of c may or may not be a leaf. If it is not a leaf, then it cannot be the ascendent of another leaf because we assume that c is a leaf with maximum order.

6. Now consider revealing the outcome of X in two steps. In the first step, if the outcome of X is not a leaf descending from node k, we identify the outcome exactly, otherwise we identify the outcome to be a child of node k. We call this random variable V.

7. If we do not identify the outcome exactly in the first step, which happens with probability q_k, we further identify in the second step which of the children (child) of node k the outcome is (there is only one child of node k which can be the outcome if all the siblings of c are not leaves). We call this random variable W.

8. If the second step is not necessary, we assume that W takes a constant value with probability 1.

9. Then $X = (V, W)$.

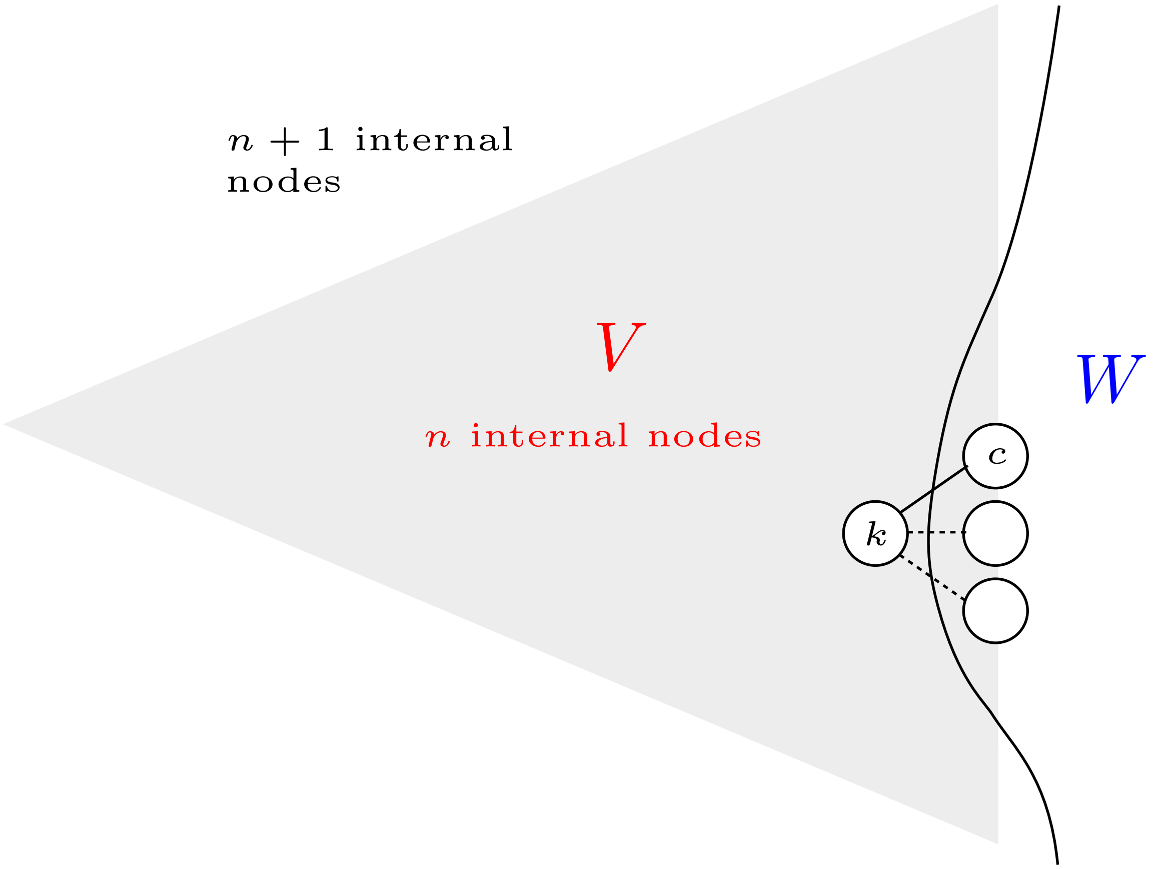
10. The outcome of V can be represented by a code tree with n internal nodes which is obtained by pruning the original code tree at node k.

11. By the induction hypothesis,

$$H(V) = \sum_{k'\in\mathcal{I}\backslash\{k\}} q_{k'} h_{k'}.$$

12. Then we have

$$\begin{aligned} H(X) &= H(V, W) \\ &= H(V) + H(W|V) \\ &= \sum_{k'\in\mathcal{I}\backslash\{k\}} q_{k'} h_{k'} + \cancel{(1-q_k)\cdot 0} + q_k h_k \end{aligned}$$



Lemma 4.19 $H_D(X) = \sum_{k\in\mathcal{I}} q_k h_k$.

Proof

1. We prove the lemma by induction on the number of internal nodes of the code tree.

2. If there is only one internal node, it must be the root of the tree. Then the lemma is trivially true upon observing that the reaching probability of the root is equal to 1.

3. Assume the lemma is true for all code trees with n internal nodes, and consider a code tree with $n+1$ internal nodes.

4. Let k be an internal node such that k is the parent of a leaf c with maximum order.

5. Each sibling of c may or may not be a leaf. If it is not a leaf, then it cannot be the ascendent of another leaf because we assume that c is a leaf with maximum order.

6. Now consider revealing the outcome of X in two steps. In the first step, if the outcome of X is not a leaf descending from node k, we identify the outcome exactly, otherwise we identify the outcome to be a child of node k. We call this random variable V.

7. If we do not identify the outcome exactly in the first step, which happens with probability q_k, we further identify in the second step which of the children (child) of node k the outcome is (there is only one child of node k which can be the outcome if all the siblings of c are not leaves). We call this random variable W.

8. If the second step is not necessary, we assume that W takes a constant value with probability 1.

9. Then $X = (V, W)$.

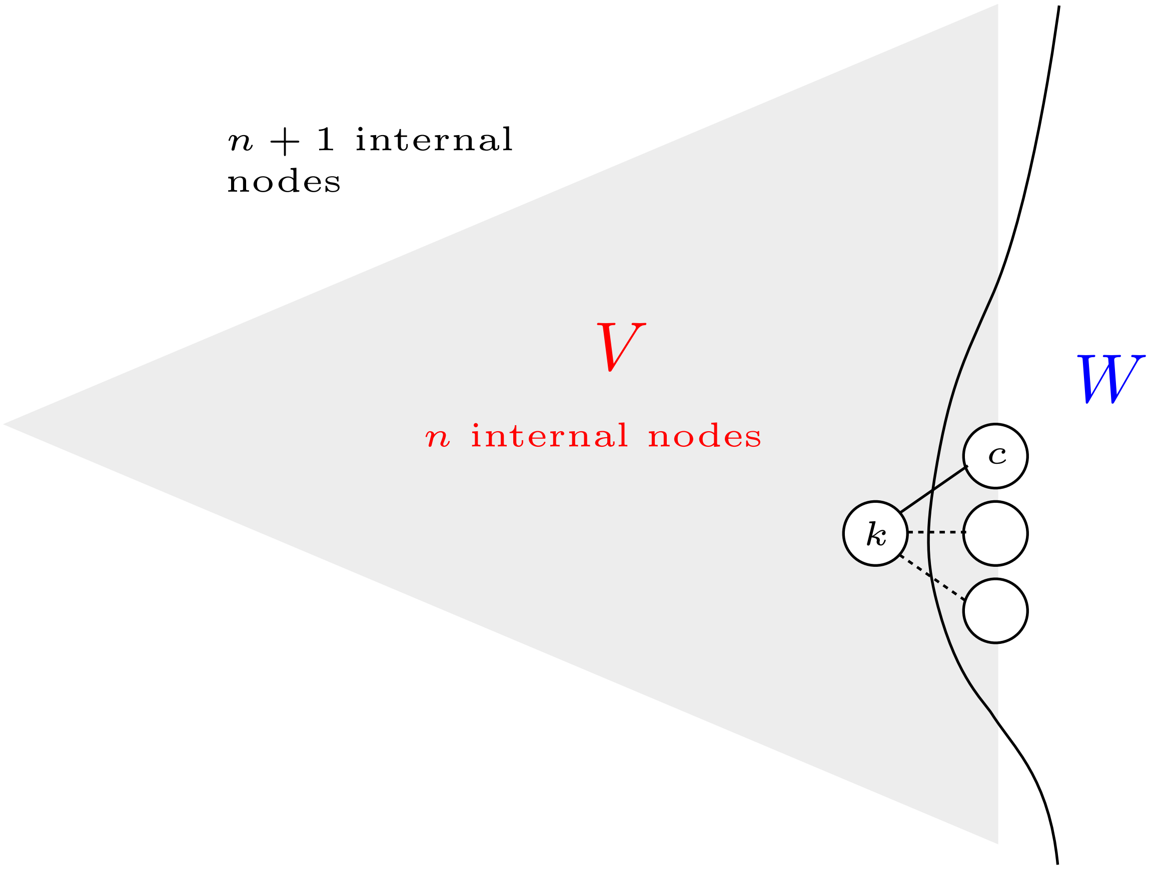
10. The outcome of V can be represented by a code tree with n internal nodes which is obtained by pruning the original code tree at node k.

11. By the induction hypothesis,

$$H(V) = \sum_{k'\in\mathcal{I}\backslash\{k\}} q_{k'} h_{k'}.$$

12. Then we have

$$\begin{aligned}
H(X) &= H(V, W) \\
&= H(V) + H(W|V) \\
&= \sum_{k'\in\mathcal{I}\backslash\{k\}} q_{k'} h_{k'} + \cancel{(1 - q_k) \cdot 0} + \underline{q_k h_k}
\end{aligned}$$



Lemma 4.19 $H_D(X) = \sum_{k \in \mathcal{I}} q_k h_k$.

Proof

1. We prove the lemma by induction on the number of internal nodes of the code tree.

2. If there is only one internal node, it must be the root of the tree. Then the lemma is trivially true upon observing that the reaching probability of the root is equal to 1.

3. Assume the lemma is true for all code trees with n internal nodes, and consider a code tree with $n+1$ internal nodes.

4. Let k be an internal node such that k is the parent of a leaf c with maximum order.

5. Each sibling of c may or may not be a leaf. If it is not a leaf, then it cannot be the ascendent of another leaf because we assume that c is a leaf with maximum order.

6. Now consider revealing the outcome of X in two steps. In the first step, if the outcome of X is not a leaf descending from node k, we identify the outcome exactly, otherwise we identify the outcome to be a child of node k. We call this random variable V.

7. If we do not identify the outcome exactly in the first step, which happens with probability q_k, we further identify in the second step which of the children (child) of node k the outcome is (there is only one child of node k which can be the outcome if all the siblings of c are not leaves). We call this random variable W.

8. If the second step is not necessary, we assume that W takes a constant value with probability 1.

9. Then $X = (V, W)$.

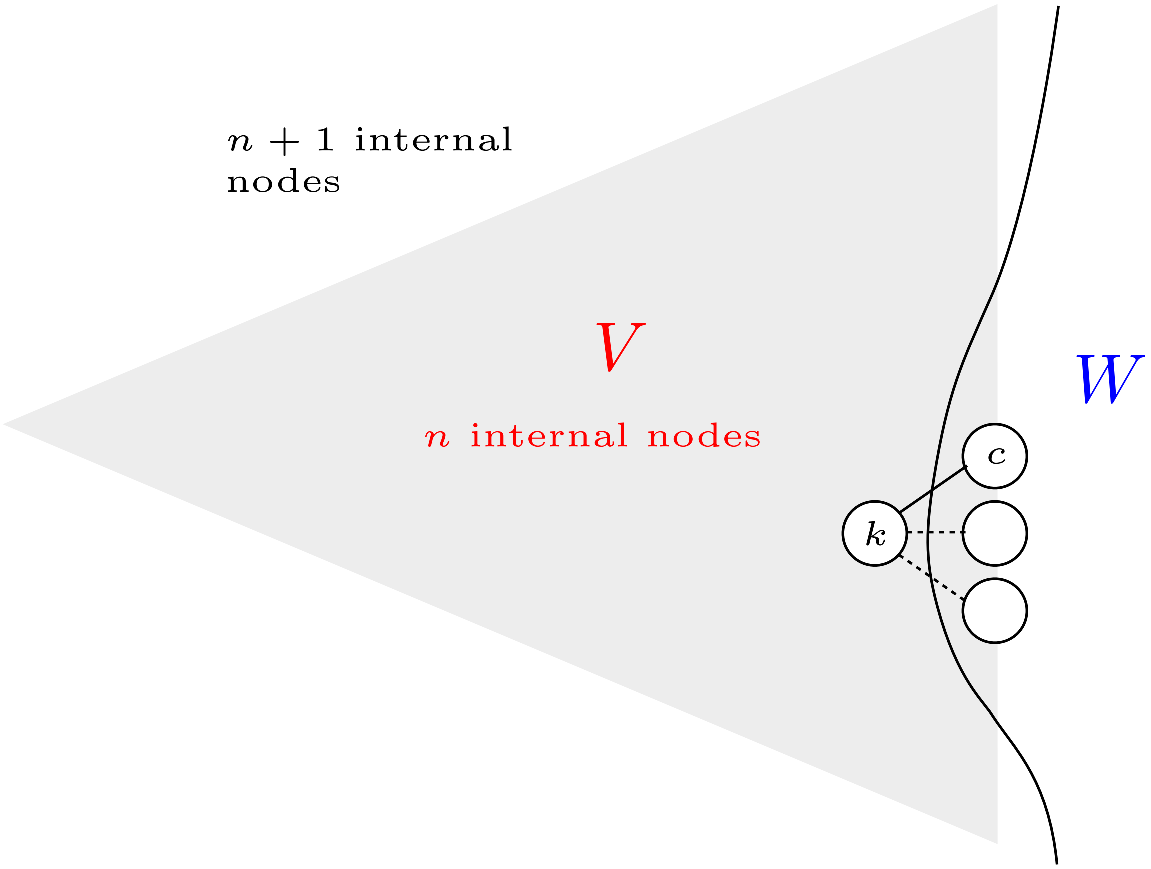
10. The outcome of V can be represented by a code tree with n internal nodes which is obtained by pruning the original code tree at node k.

11. By the induction hypothesis,

$$H(V) = \sum_{k' \in \mathcal{I} \setminus \{k\}} q_{k'} h_{k'}.$$

12. Then we have

$$\begin{aligned} H(X) &= H(V, W) \\ &= H(V) + H(W|V) \\ &= \sum_{k' \in \mathcal{I} \setminus \{k\}} \underline{q_{k'} h_{k'}} + \cancel{(1 - q_k) \cdot 0} + \underline{q_k h_k} \end{aligned}$$



Lemma 4.19 $H_D(X) = \sum_{k \in \mathcal{I}} q_k h_k$.

Proof

1. We prove the lemma by induction on the number of internal nodes of the code tree.

2. If there is only one internal node, it must be the root of the tree. Then the lemma is trivially true upon observing that the reaching probability of the root is equal to 1.

3. Assume the lemma is true for all code trees with n internal nodes, and consider a code tree with $n+1$ internal nodes.

4. Let k be an internal node such that k is the parent of a leaf c with maximum order.

5. Each sibling of c may or may not be a leaf. If it is not a leaf, then it cannot be the ascendent of another leaf because we assume that c is a leaf with maximum order.

6. Now consider revealing the outcome of X in two steps. In the first step, if the outcome of X is not a leaf descending from node k, we identify the outcome exactly, otherwise we identify the outcome to be a child of node k. We call this random variable V.

7. If we do not identify the outcome exactly in the first step, which happens with probability q_k, we further identify in the second step which of the children (child) of node k the outcome is (there is only one child of node k which can be the outcome if all the siblings of c are not leaves). We call this random variable W.

8. If the second step is not necessary, we assume that W takes a constant value with probability 1.

9. Then $X = (V, W)$.

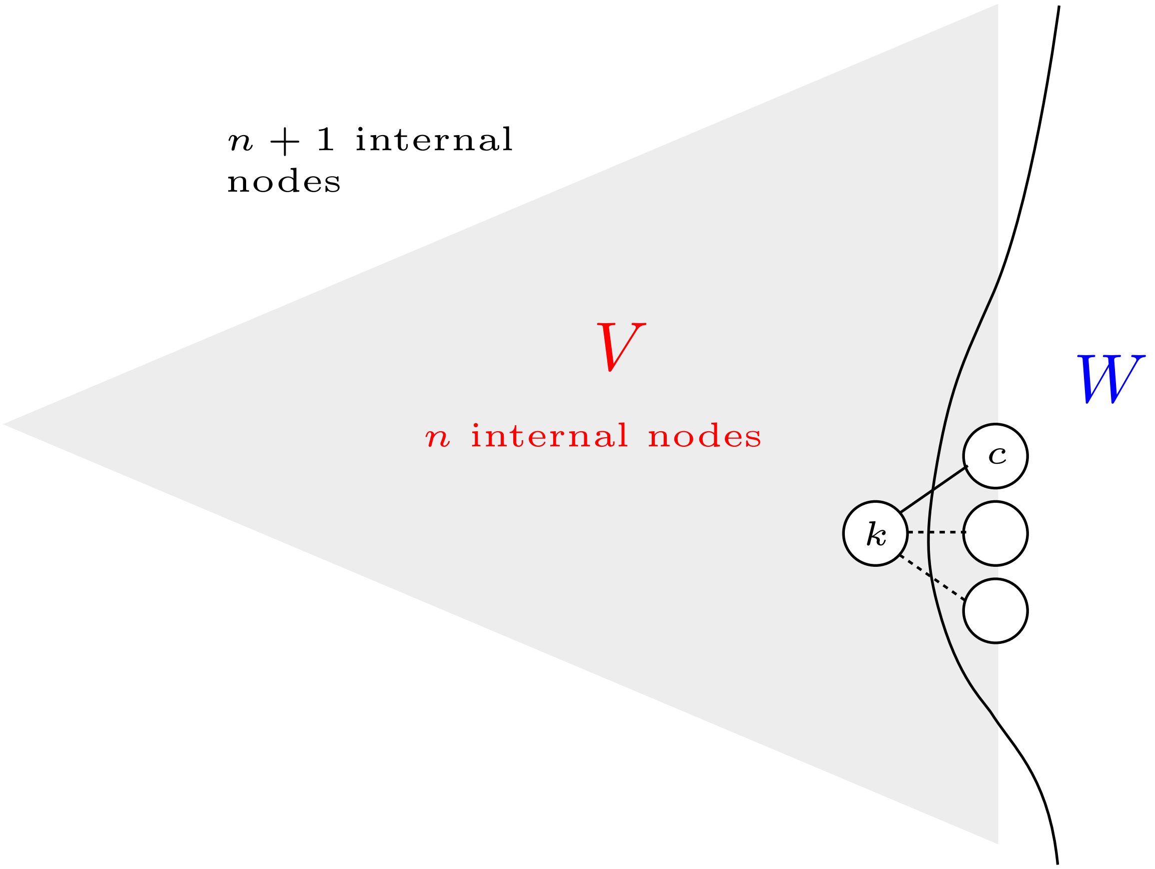
10. The outcome of V can be represented by a code tree with n internal nodes which is obtained by pruning the original code tree at node k.

11. By the induction hypothesis,

$$H(V) = \sum_{k' \in \mathcal{I} \backslash \{k\}} q_{k'} h_{k'}.$$

12. Then we have

$$\begin{aligned} H(X) &= H(V, W) \\ &= H(V) + H(W|V) \\ &= \sum_{k' \in \mathcal{I} \backslash \{k\}} \underline{q_{k'} h_{k'}} + \cancel{(1 - q_k) \cdot 0} + \underline{q_k h_k} \\ &= \sum_{\underline{k' \in \mathcal{I}}} q_{k'} h_{k'}. \end{aligned}$$



Lemma 4.19 $H_D(X) = \sum_{k \in \mathcal{I}} q_k h_k$.

Proof

1. We prove the lemma by induction on the number of internal nodes of the code tree.

2. If there is only one internal node, it must be the root of the tree. Then the lemma is trivially true upon observing that the reaching probability of the root is equal to 1.

3. Assume the lemma is true for all code trees with n internal nodes, and consider a code tree with $n+1$ internal nodes.

4. Let k be an internal node such that k is the parent of a leaf c with maximum order.

5. Each sibling of c may or may not be a leaf. If it is not a leaf, then it cannot be the ascendent of another leaf because we assume that c is a leaf with maximum order.

6. Now consider revealing the outcome of X in two steps. In the first step, if the outcome of X is not a leaf descending from node k, we identify the outcome exactly, otherwise we identify the outcome to be a child of node k. We call this random variable V.

7. If we do not identify the outcome exactly in the first step, which happens with probability q_k, we further identify in the second step which of the children (child) of node k the outcome is (there is only one child of node k which can be the outcome if all the siblings of c are not leaves). We call this random variable W.

8. If the second step is not necessary, we assume that W takes a constant value with probability 1.

9. Then $X = (V, W)$.

10. The outcome of V can be represented by a code tree with n internal nodes which is obtained by pruning the original code tree at node k.

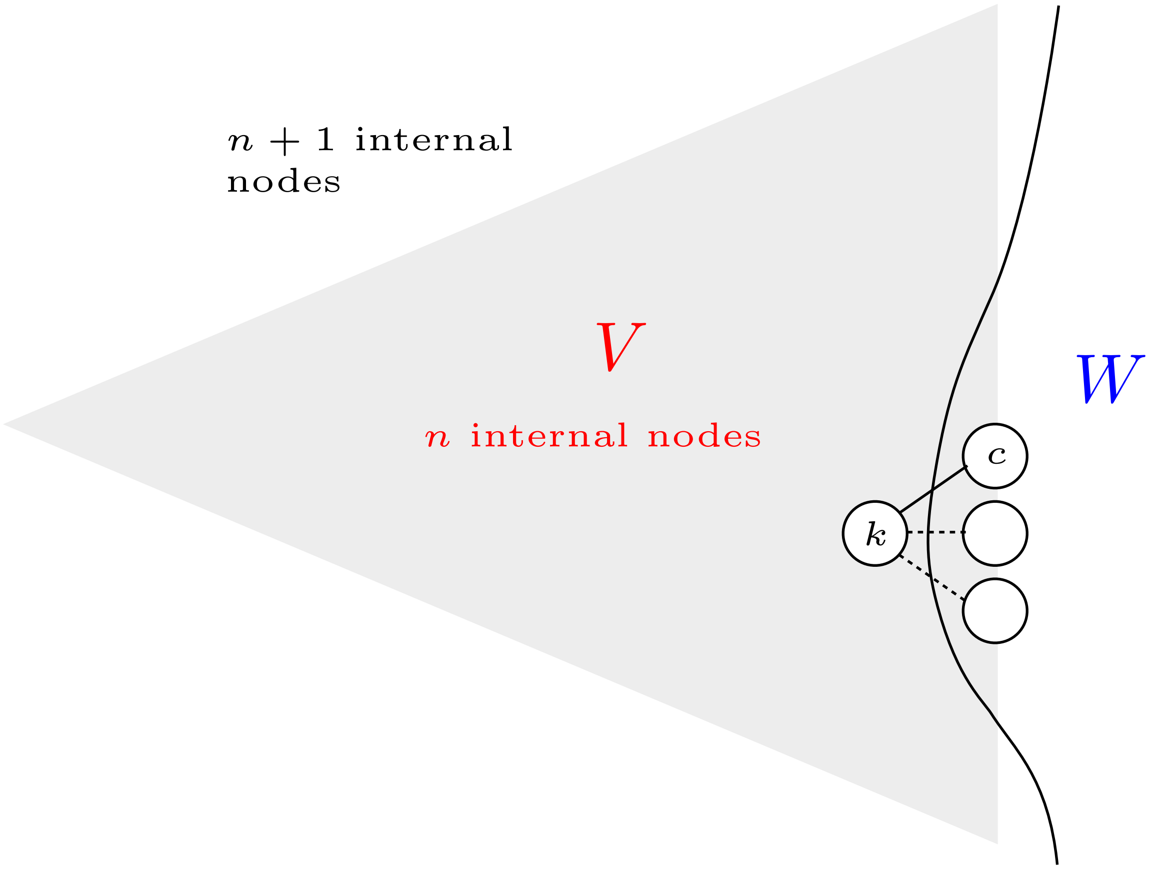
11. By the induction hypothesis,

$$H(V) = \sum_{k' \in \mathcal{I} \backslash \{k\}} q_{k'} h_{k'}.$$

12. Then we have

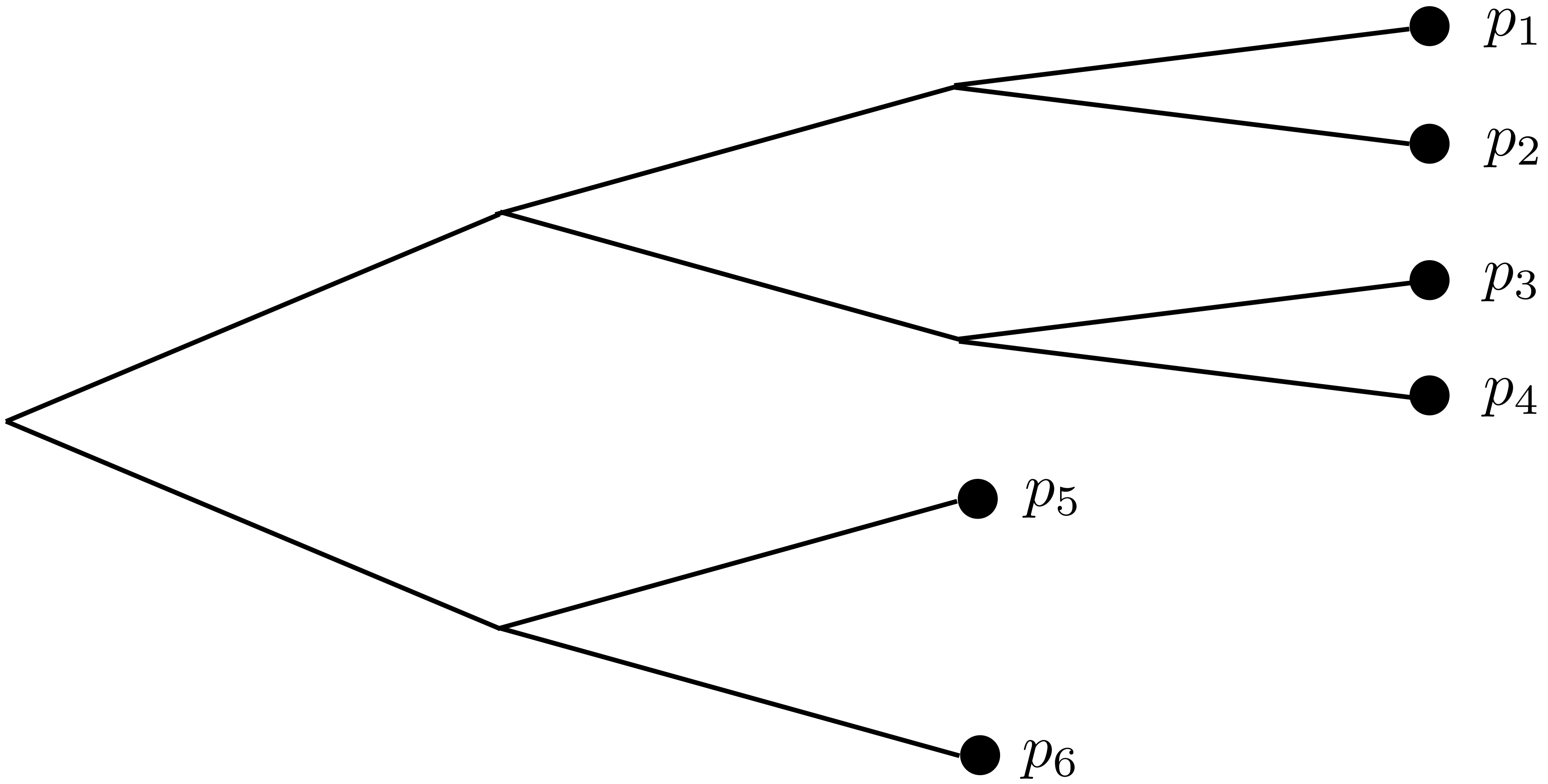
$$\begin{aligned} H(X) &= H(V, W) \\ &= H(V) + H(W|V) \\ &= \sum_{k' \in \mathcal{I} \backslash \{k\}} q_{k'} h_{k'} + (1 - q_k) \cdot 0 + q_k h_k \\ &= \sum_{k' \in \mathcal{I}} q_{k'} h_{k'}. \end{aligned}$$

The lemma is proved.



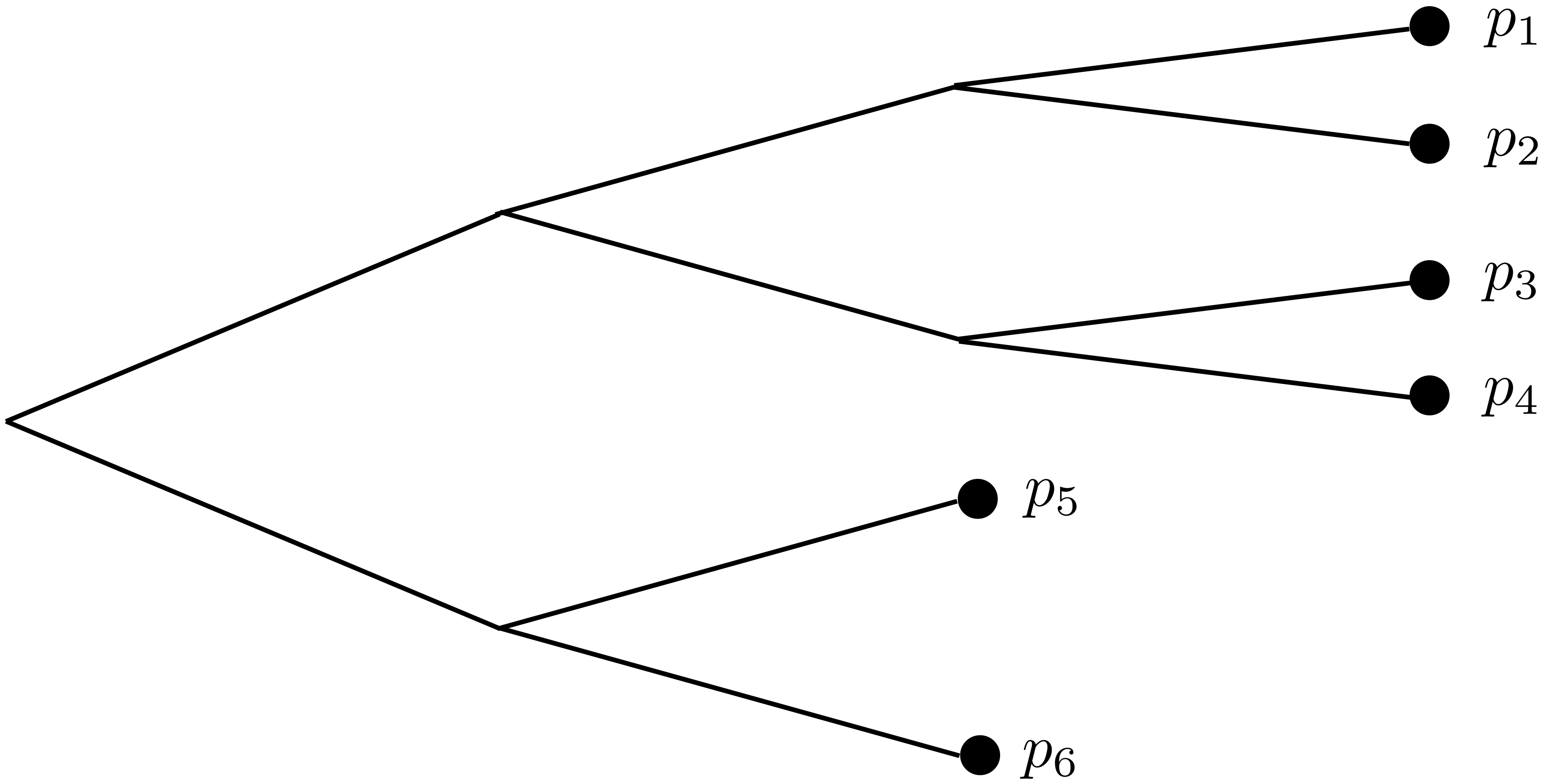
Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.



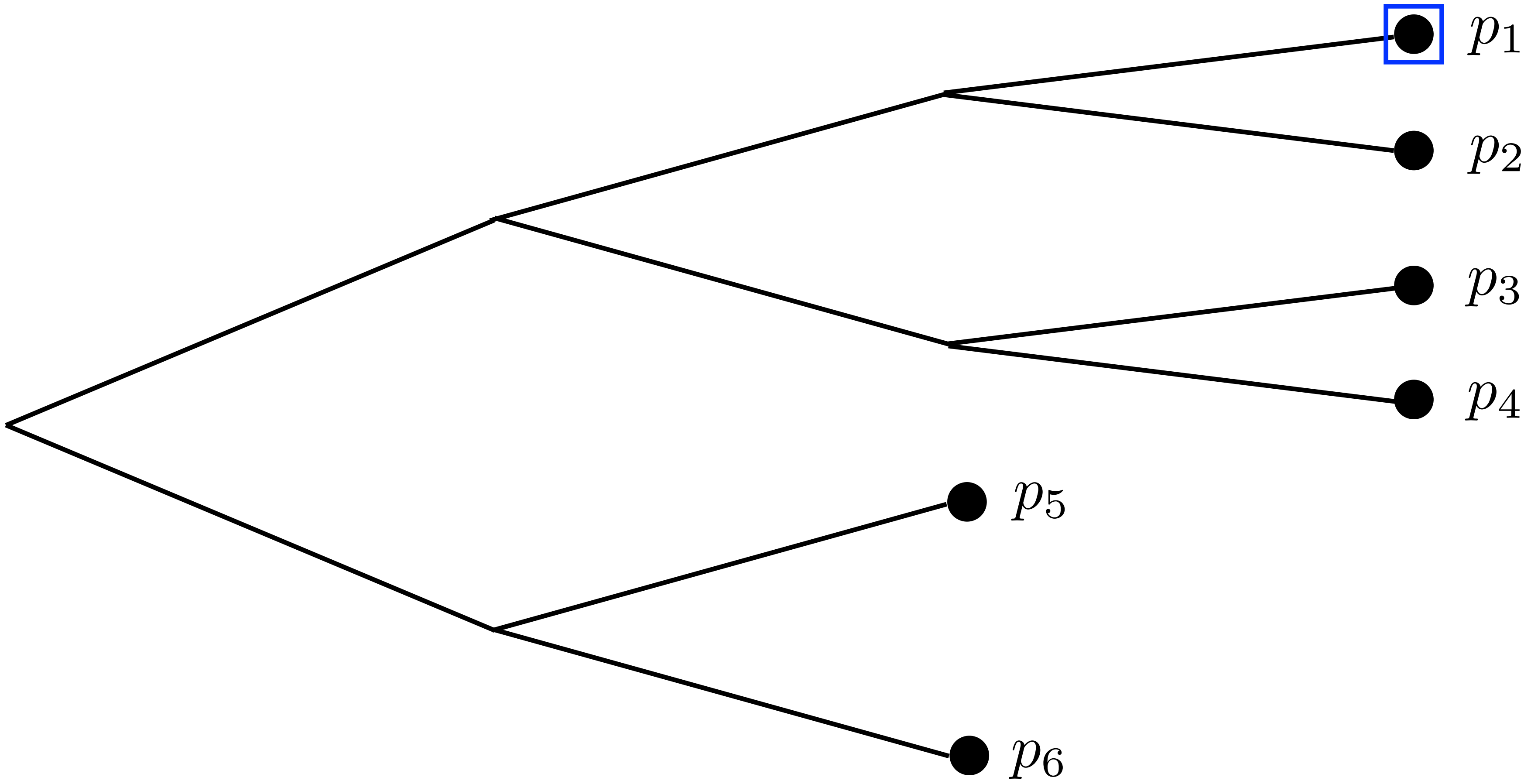
Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

$$\boxed{L = 3p_1 + 3p_2 + 3p_3 + 3p_4 + 2p_5 + 2p_6}$$



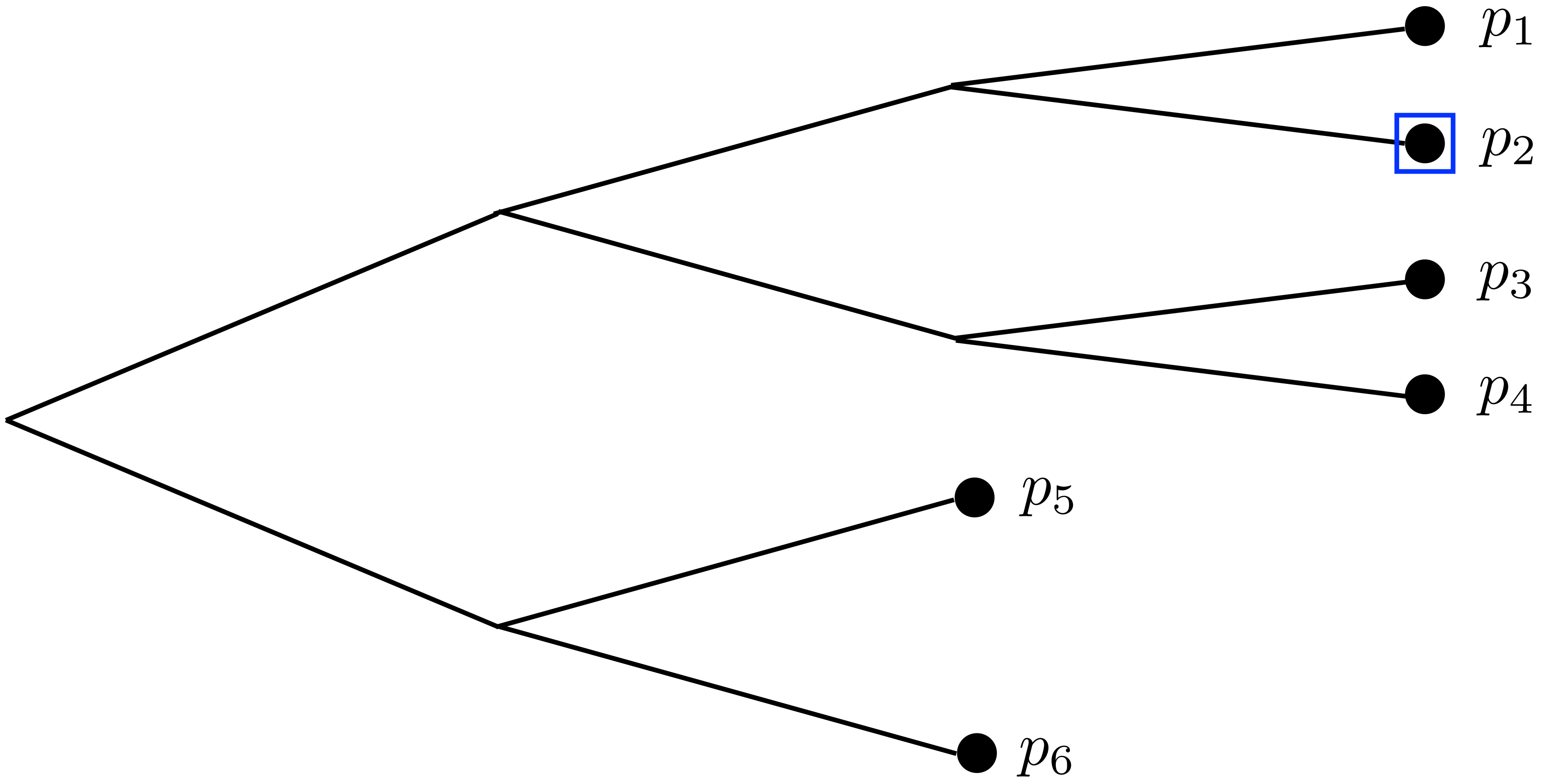
Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

$$\boxed{L = \underline{3p_1} + 3p_2 + 3p_3 + 3p_4 + 2p_5 + 2p_6}$$



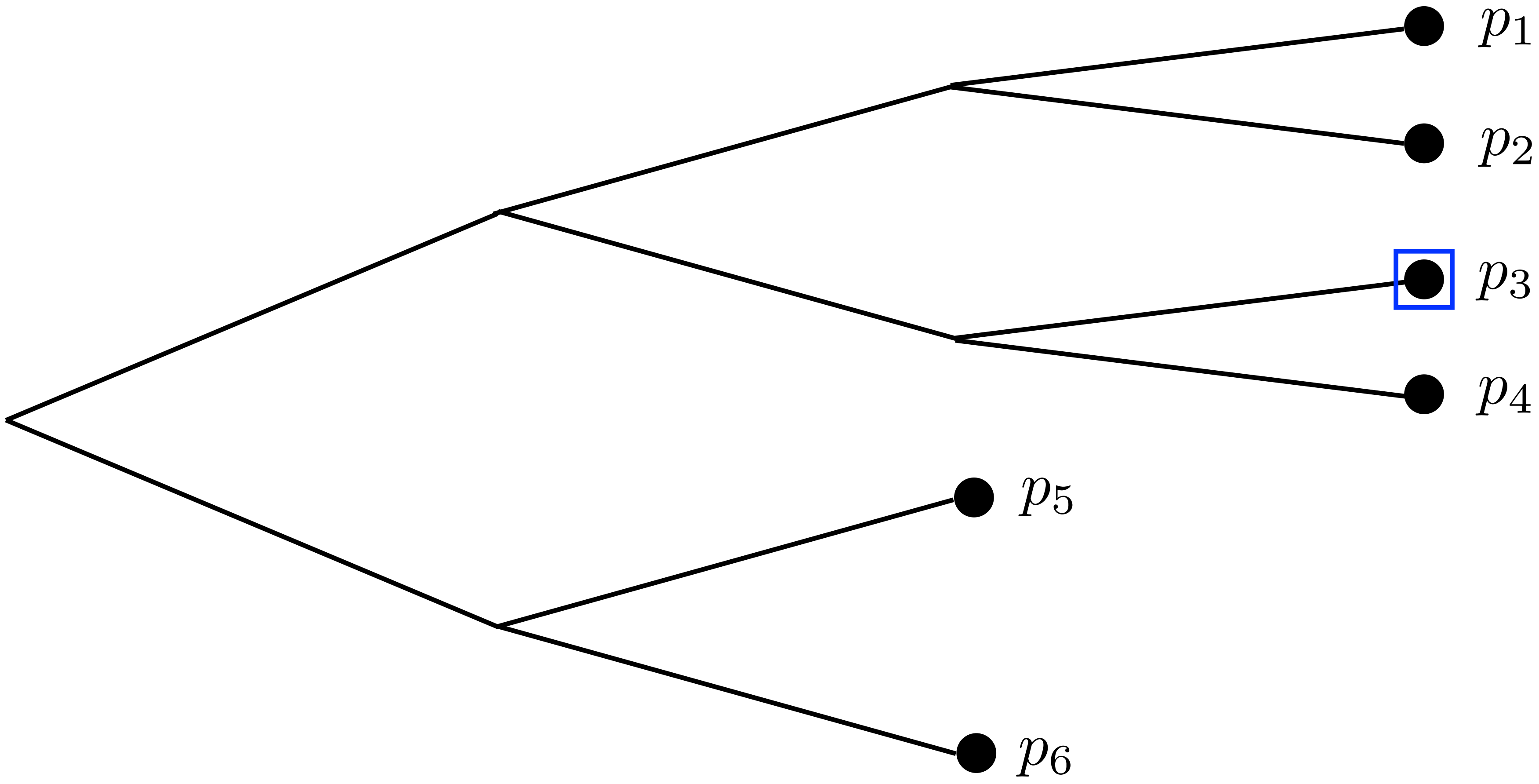
Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

$$\boxed{L = 3p_1 + \underline{3p_2} + 3p_3 + 3p_4 + 2p_5 + 2p_6}$$



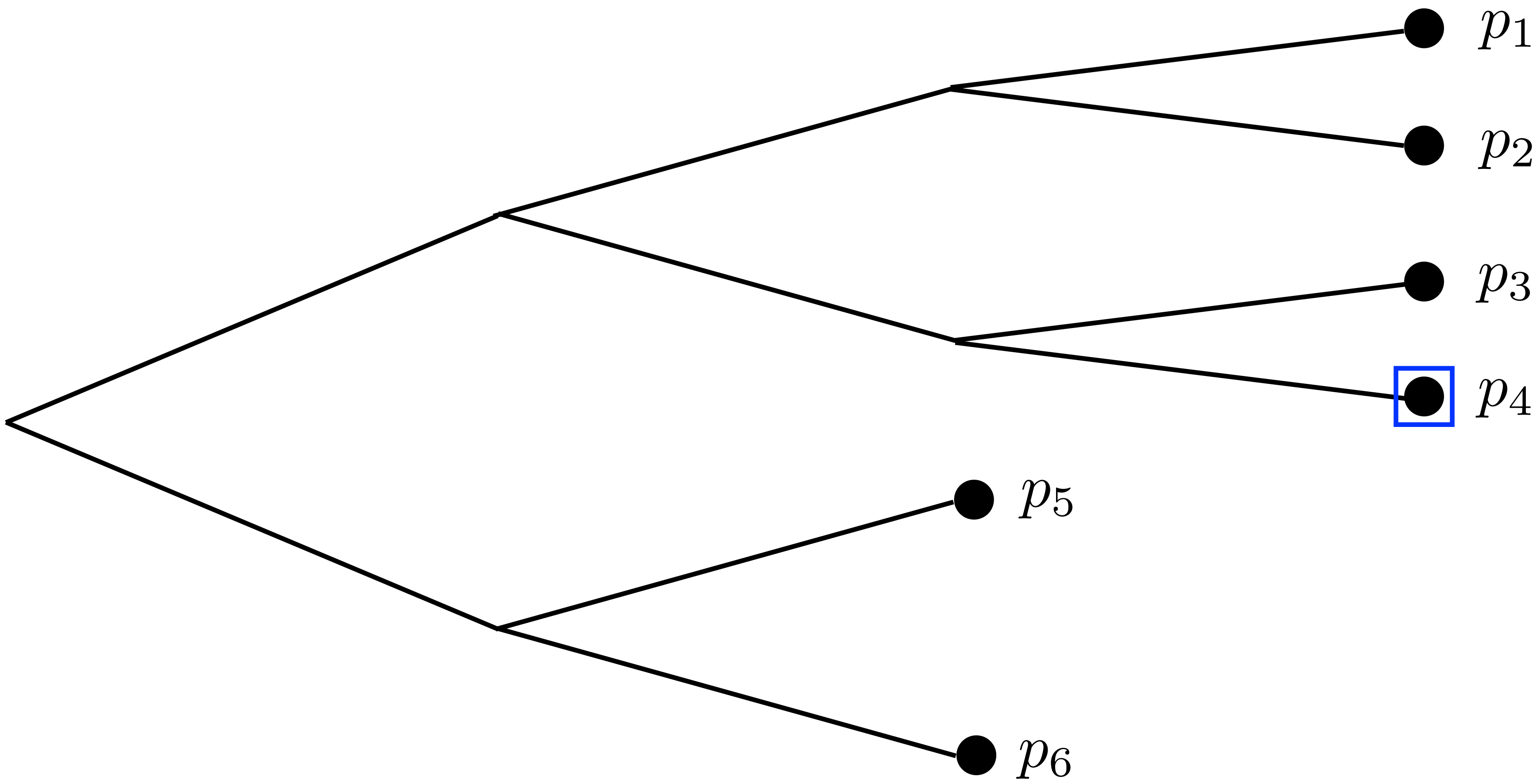
Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

$$\boxed{L = 3p_1 + 3p_2 + \underline{3p_3} + 3p_4 + 2p_5 + 2p_6}$$



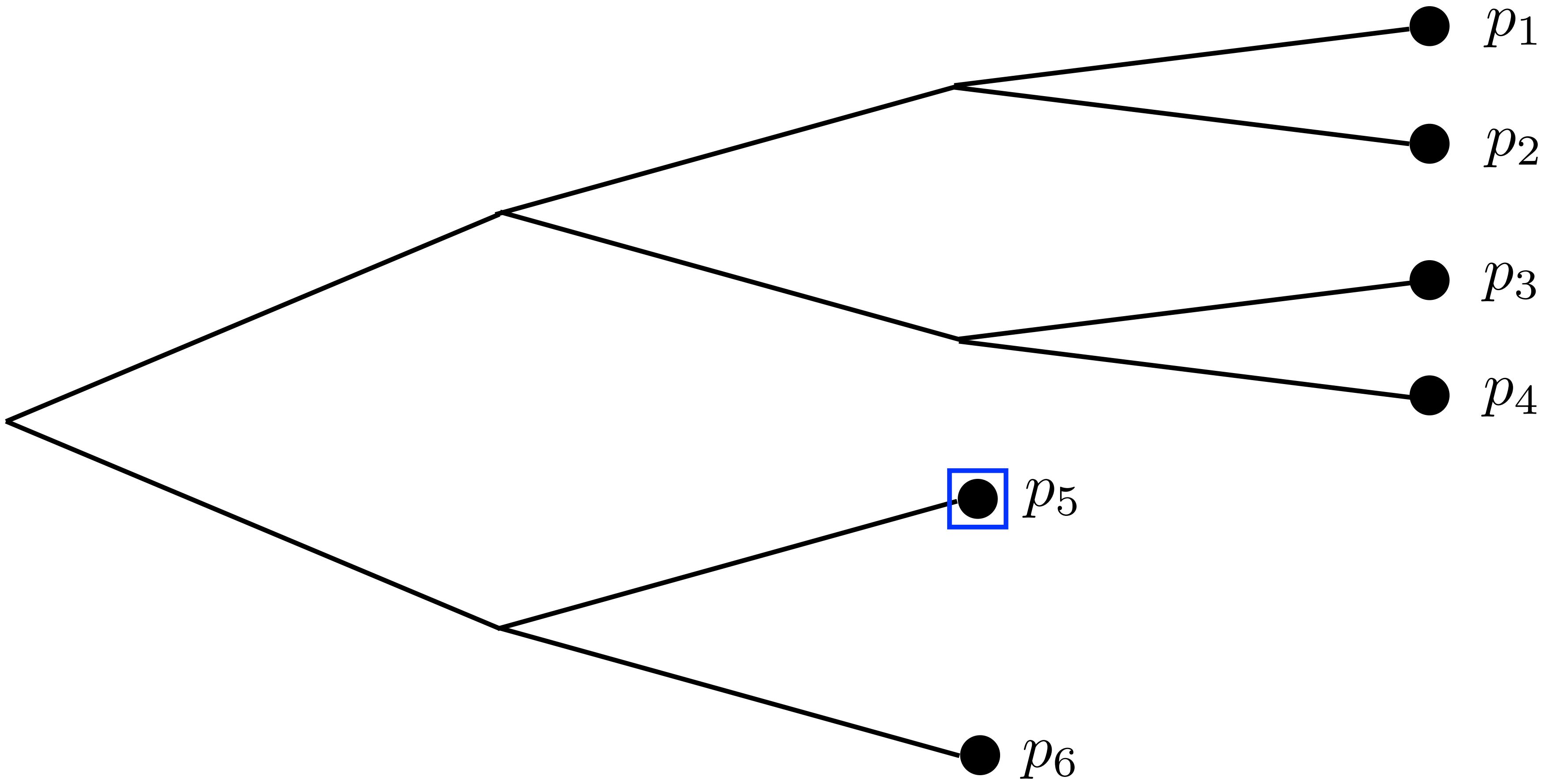
Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

$$\boxed{L = 3p_1 + 3p_2 + 3p_3 + \underline{3p_4} + 2p_5 + 2p_6}$$



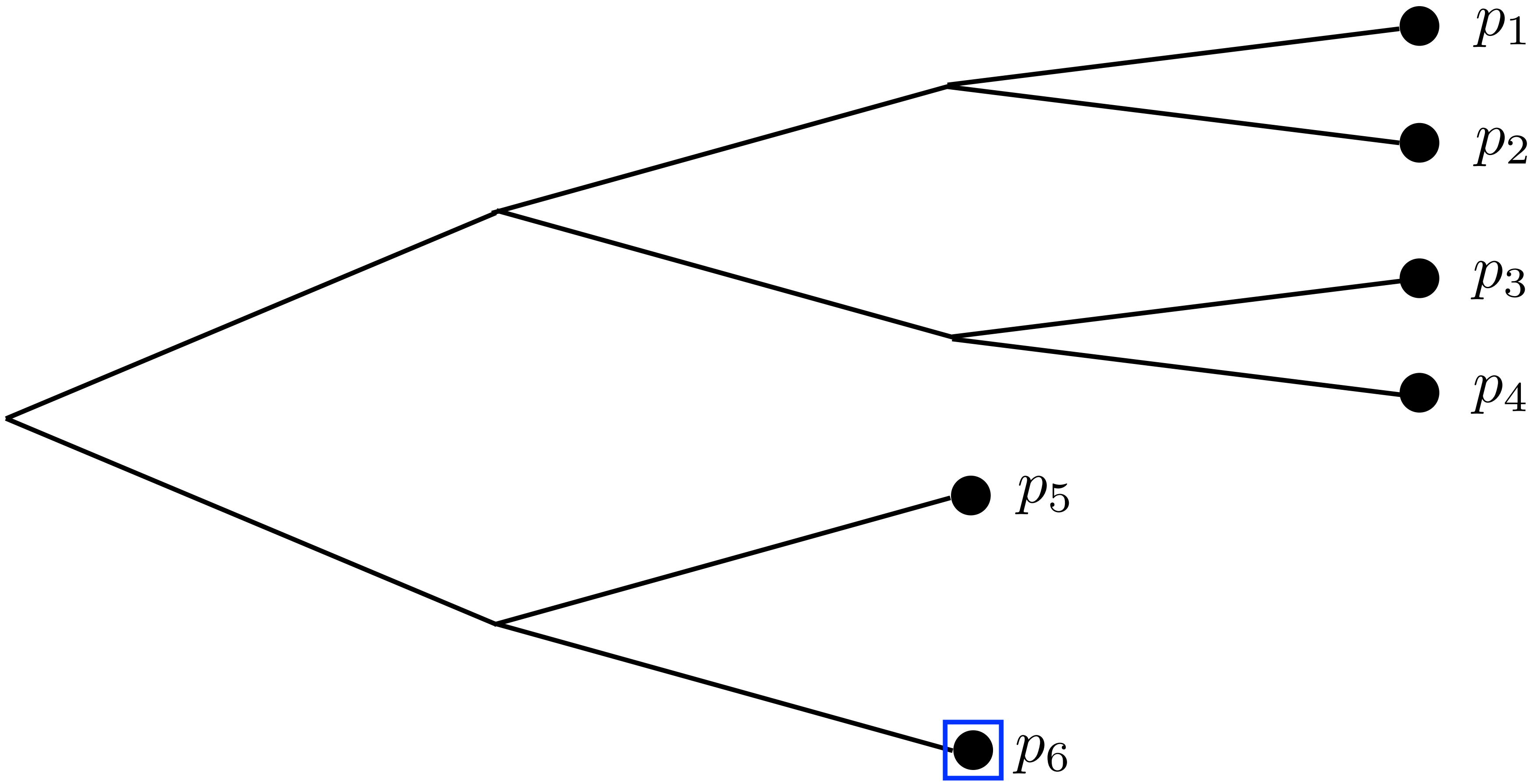
Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

$$\boxed{L = 3p_1 + 3p_2 + 3p_3 + 3p_4 + \underline{2p_5} + 2p_6}$$



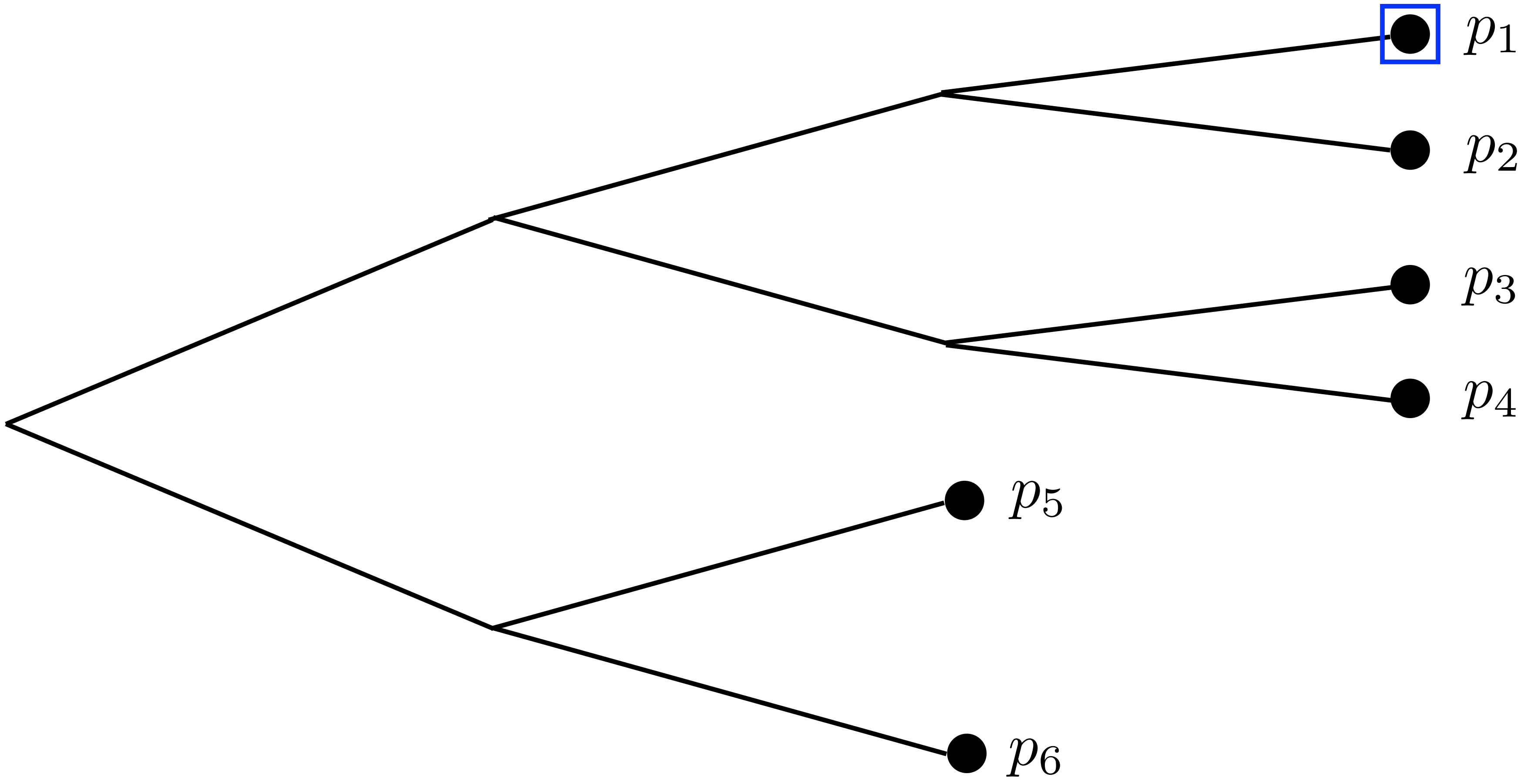
Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

$$\boxed{L = 3p_1 + 3p_2 + 3p_3 + 3p_4 + 2p_5 + \underline{2p_6}}$$



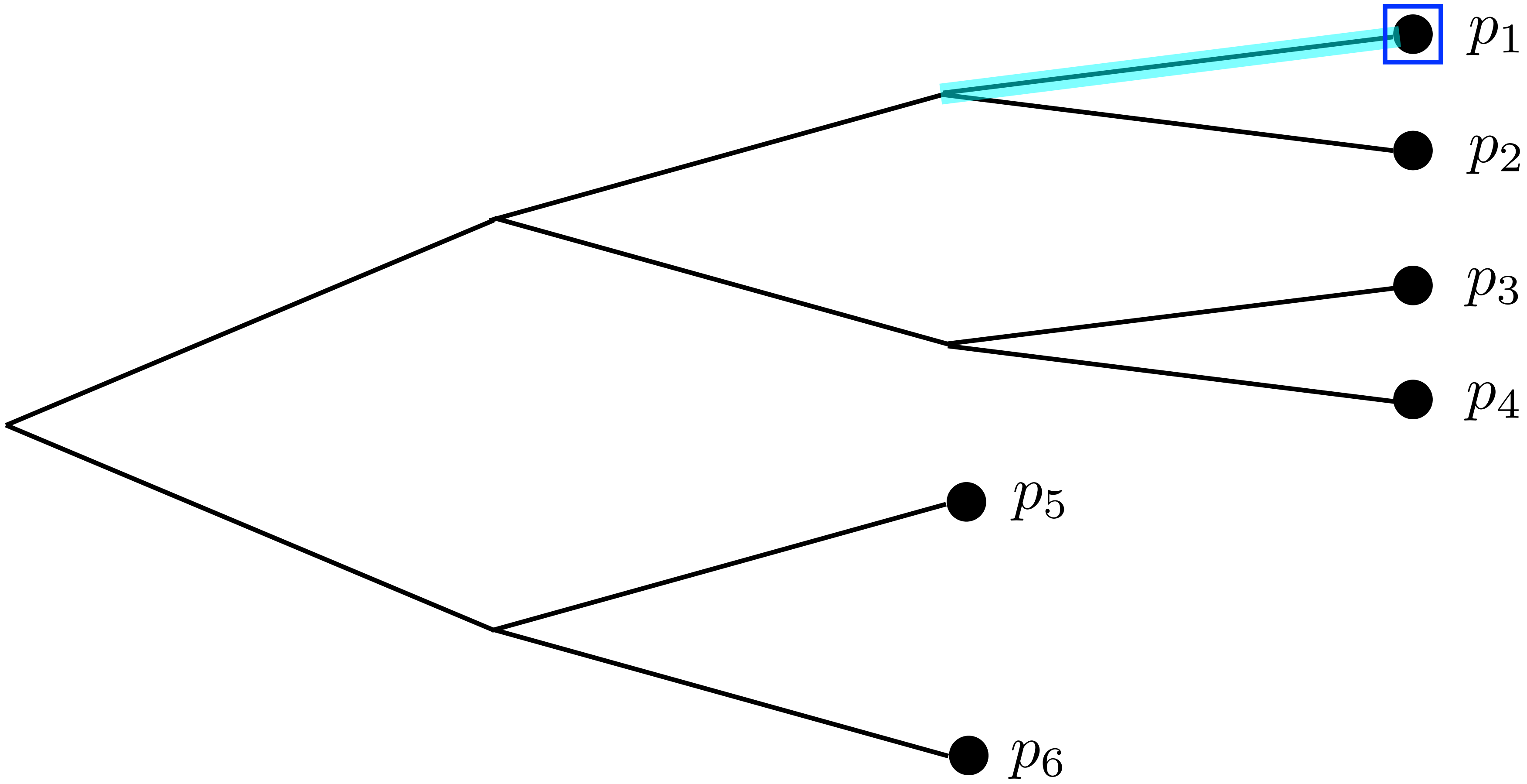
Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

$$\boxed{L = 3p_1 + 3p_2 + 3p_3 + 3p_4 + 2p_5 + 2p_6}$$



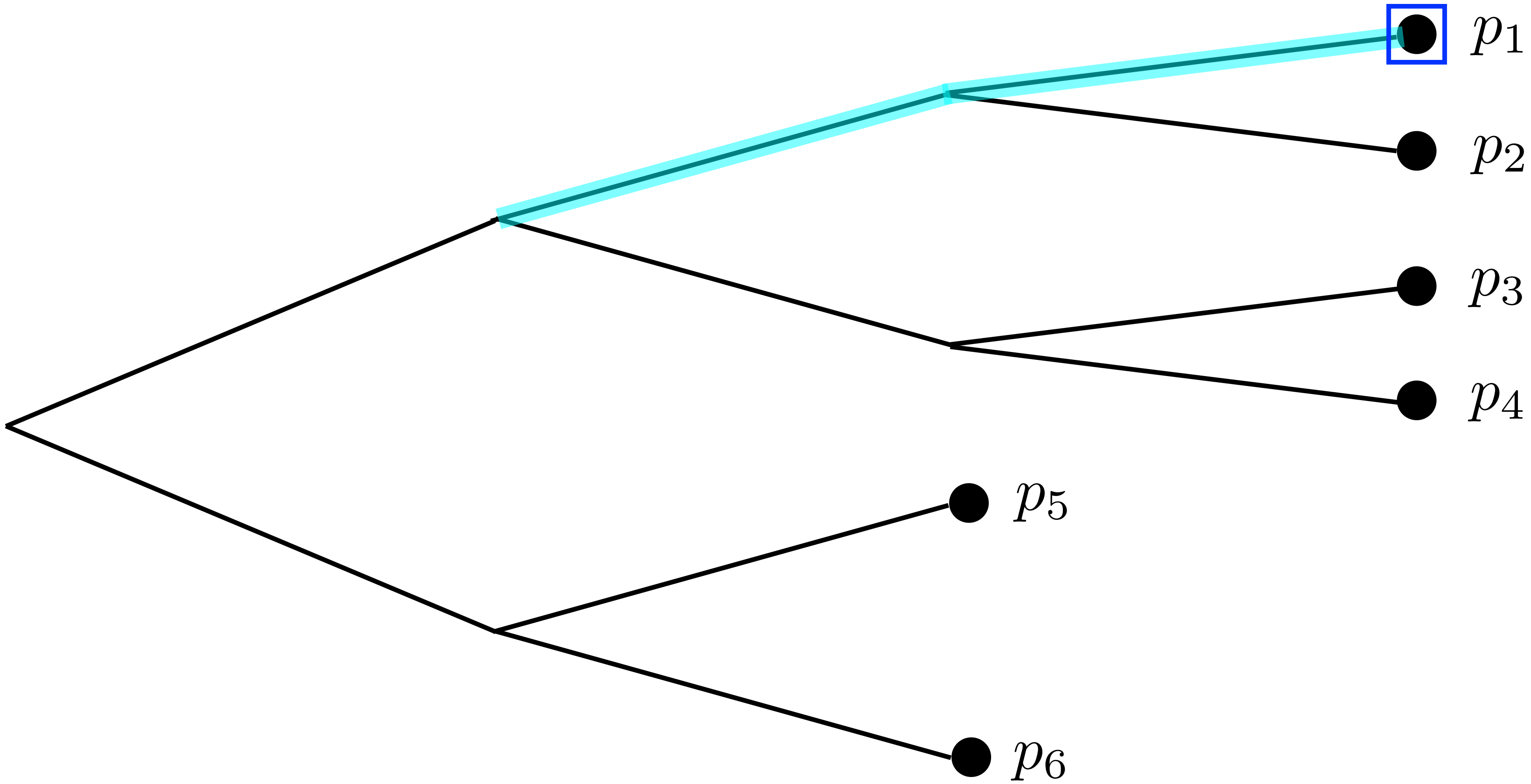
Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

$$\boxed{L = 3p_1 + 3p_2 + 3p_3 + 3p_4 + 2p_5 + 2p_6}$$



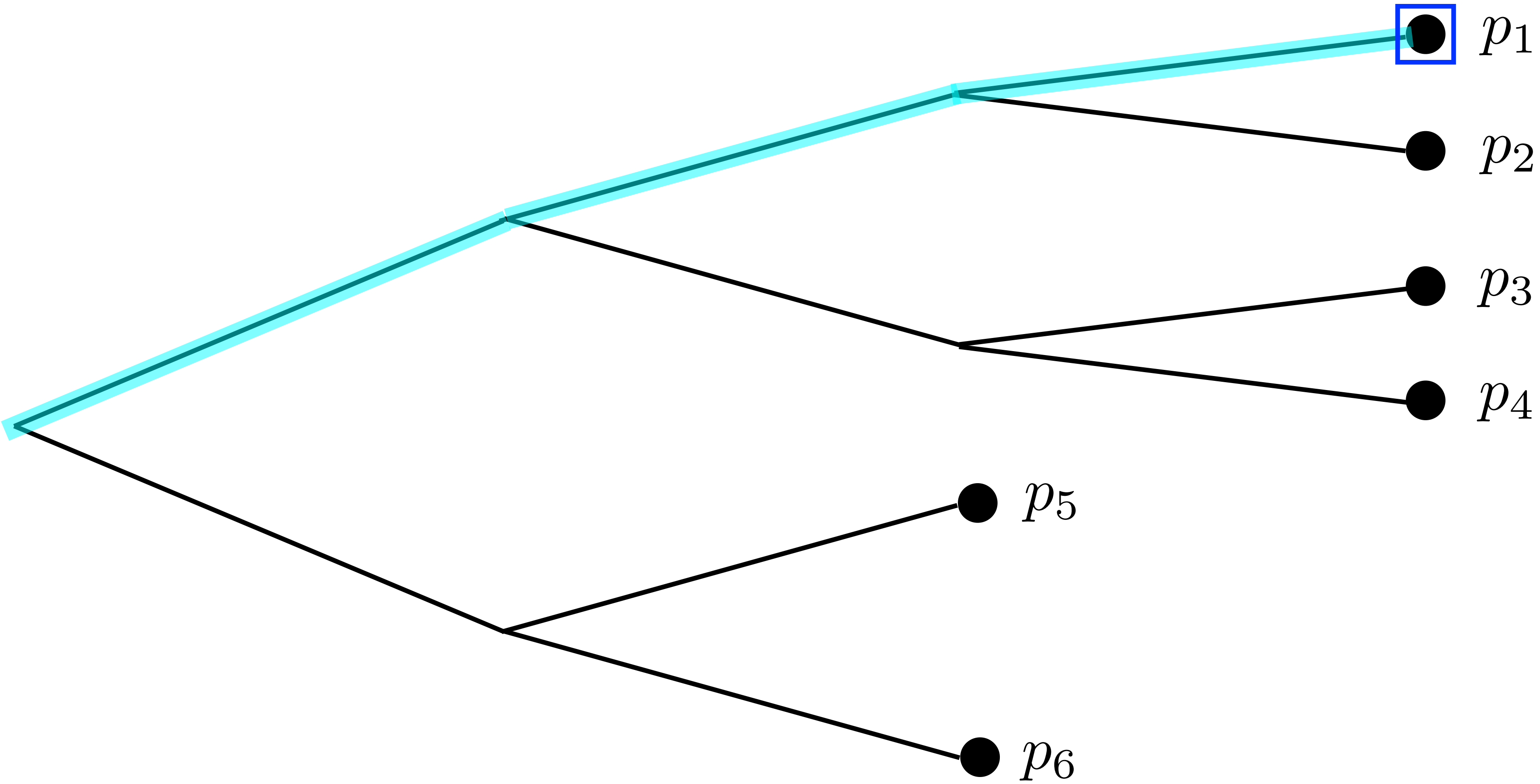
Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

$$\boxed{L = 3p_1 + 3p_2 + 3p_3 + 3p_4 + 2p_5 + 2p_6}$$



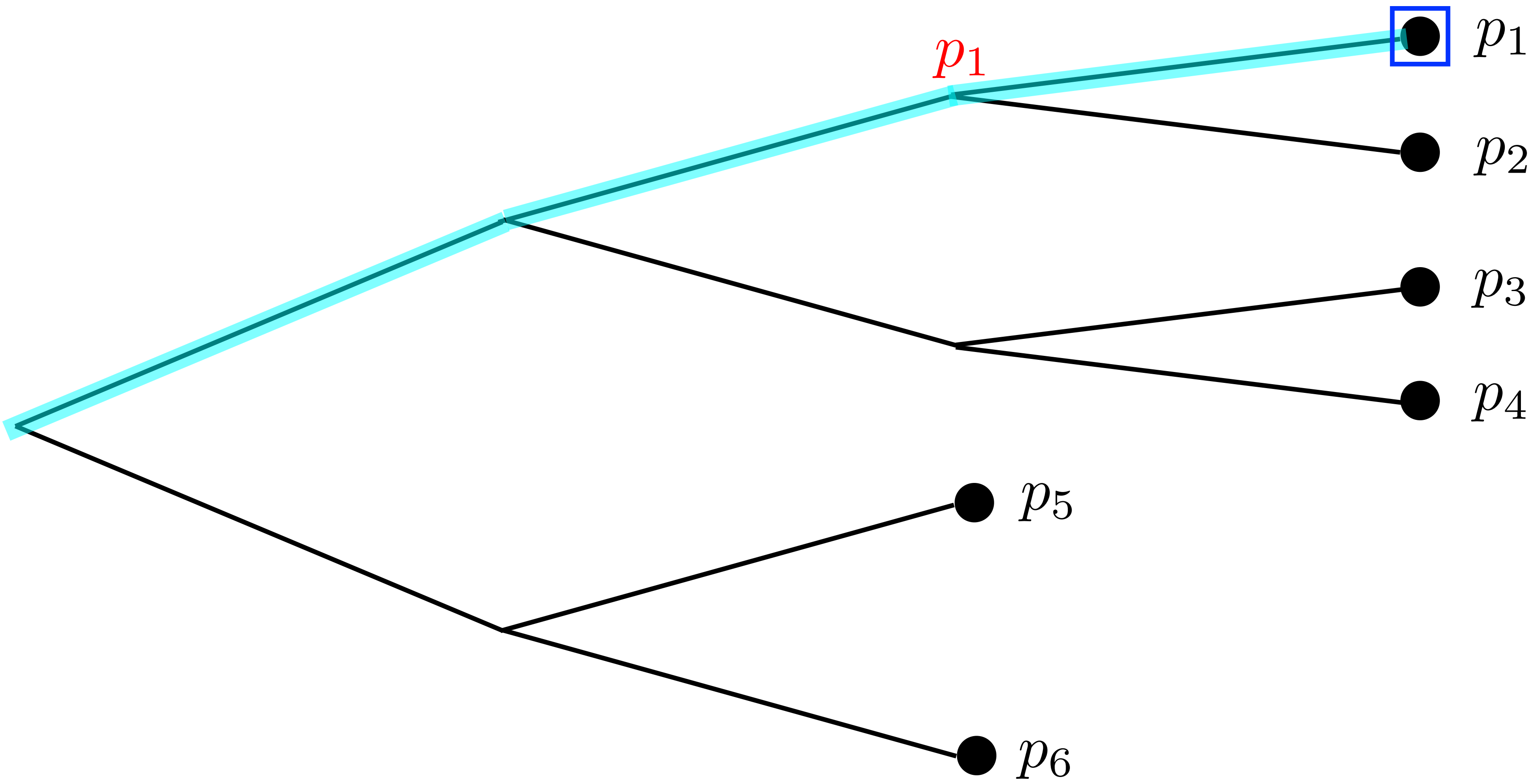
Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

$$\boxed{L = 3p_1 + 3p_2 + 3p_3 + 3p_4 + 2p_5 + 2p_6}$$



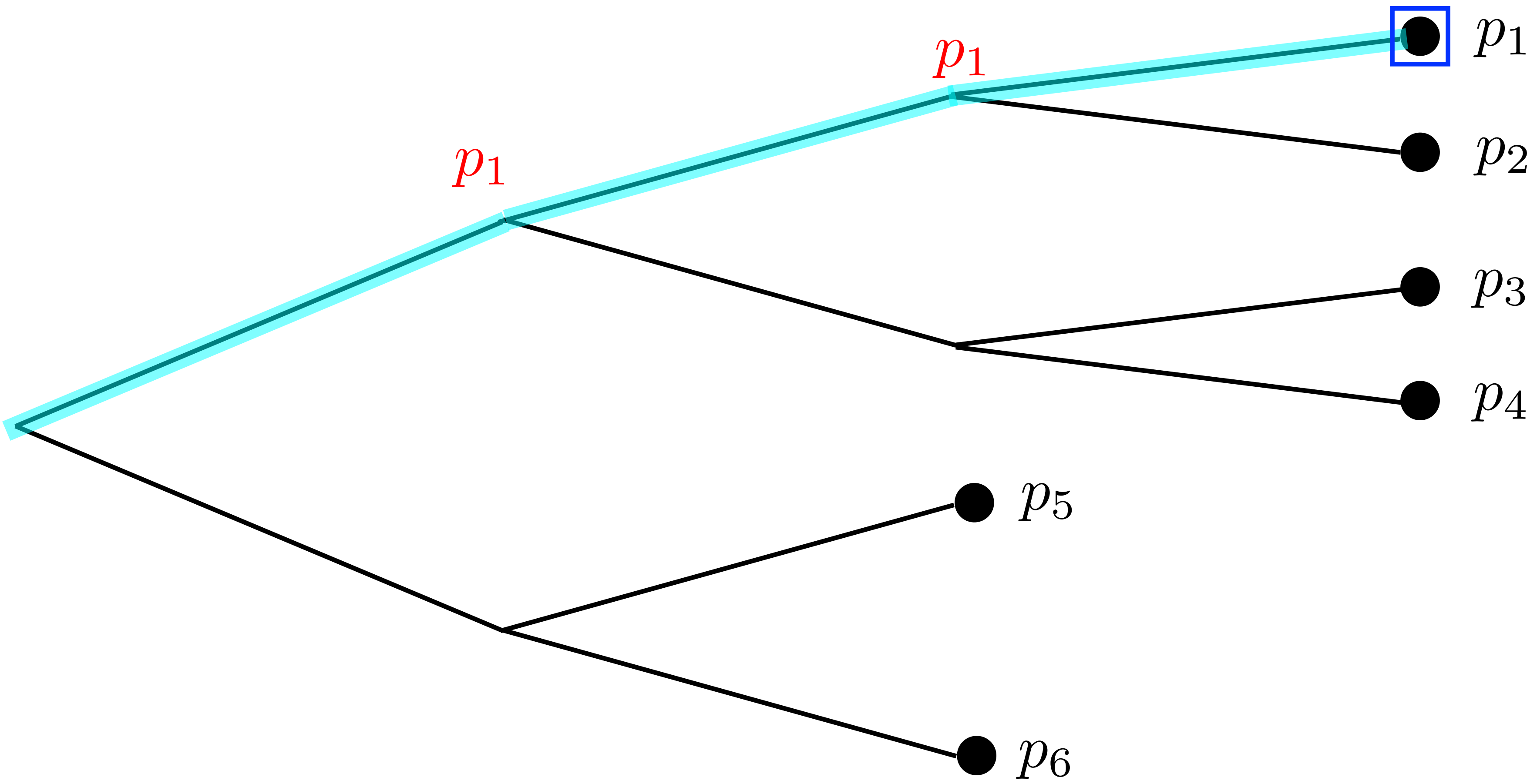
Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

$$\boxed{L = 3p_1 + 3p_2 + 3p_3 + 3p_4 + 2p_5 + 2p_6}$$



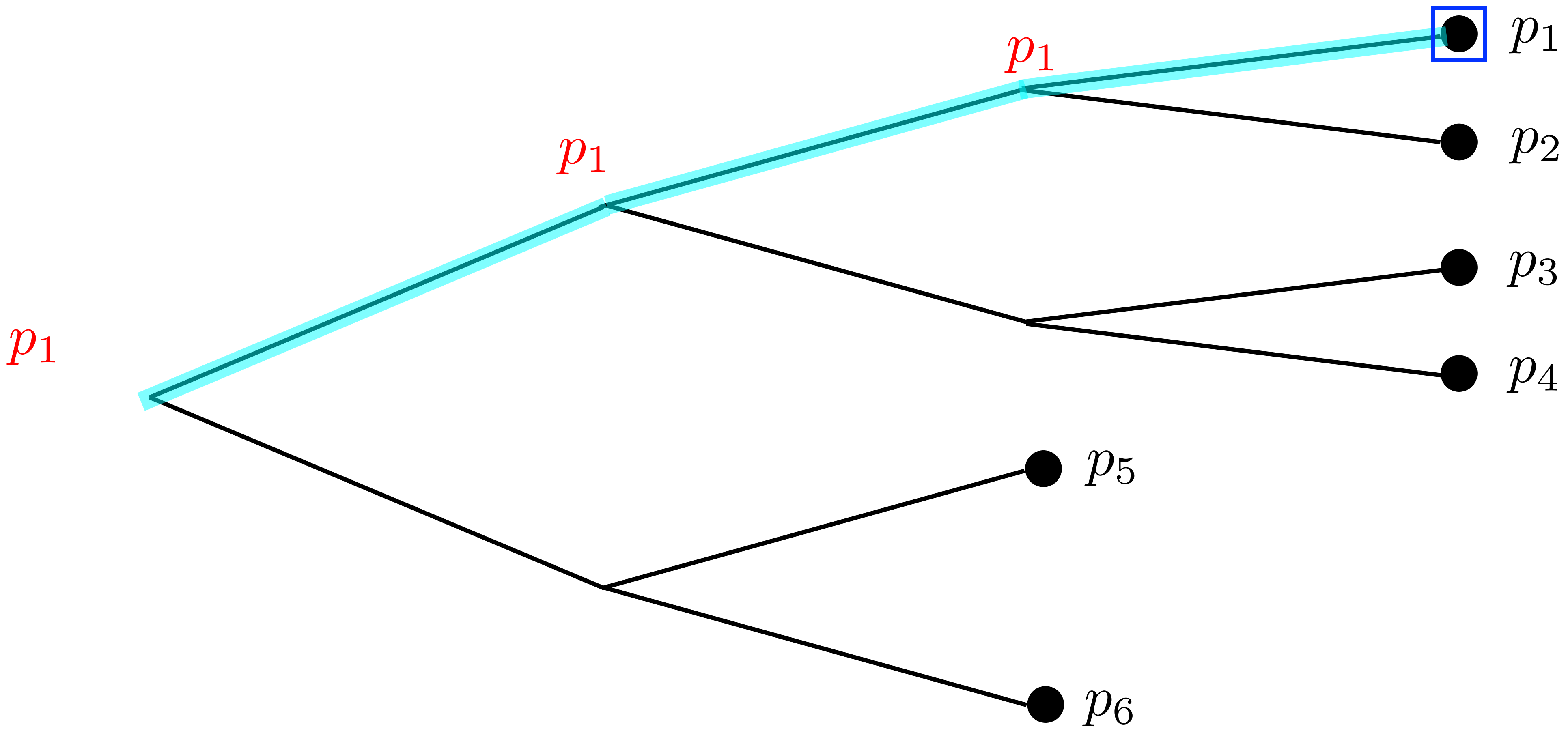
Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

$$\boxed{L = 3p_1 + 3p_2 + 3p_3 + 3p_4 + 2p_5 + 2p_6}$$



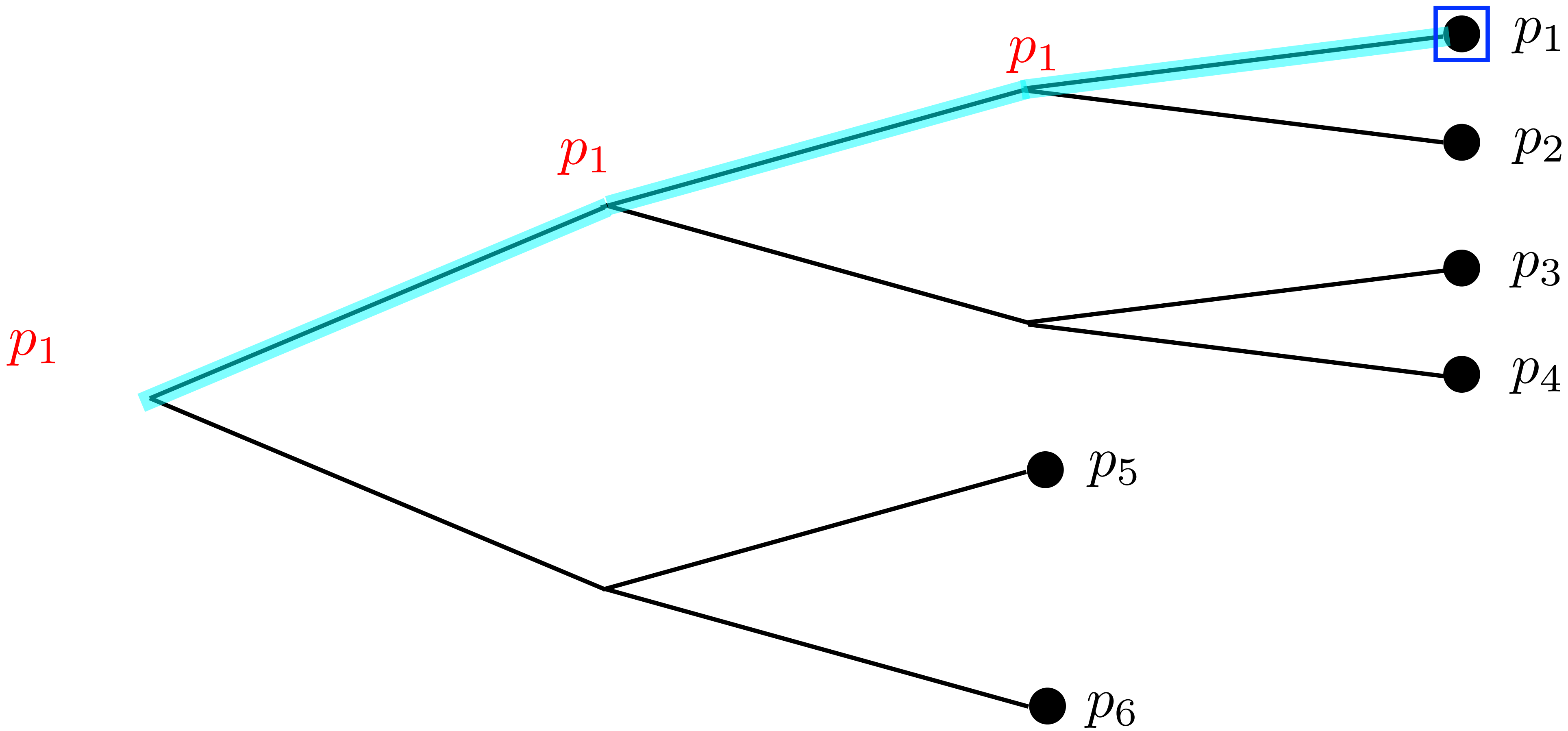
Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

$$\boxed{L = 3p_1 + 3p_2 + 3p_3 + 3p_4 + 2p_5 + 2p_6}$$



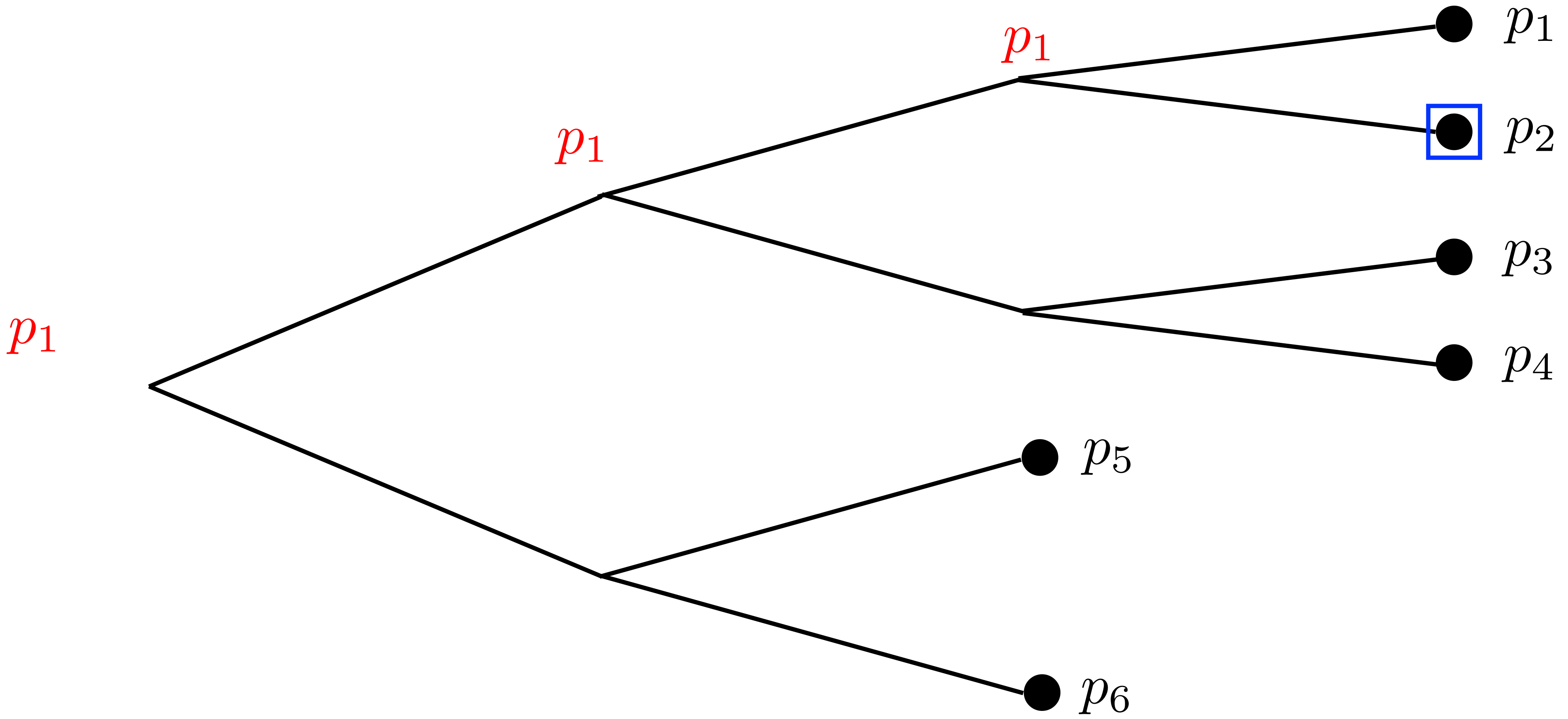
Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

$$\boxed{L = \underline{3p_1} + 3p_2 + 3p_3 + 3p_4 + 2p_5 + 2p_6}$$



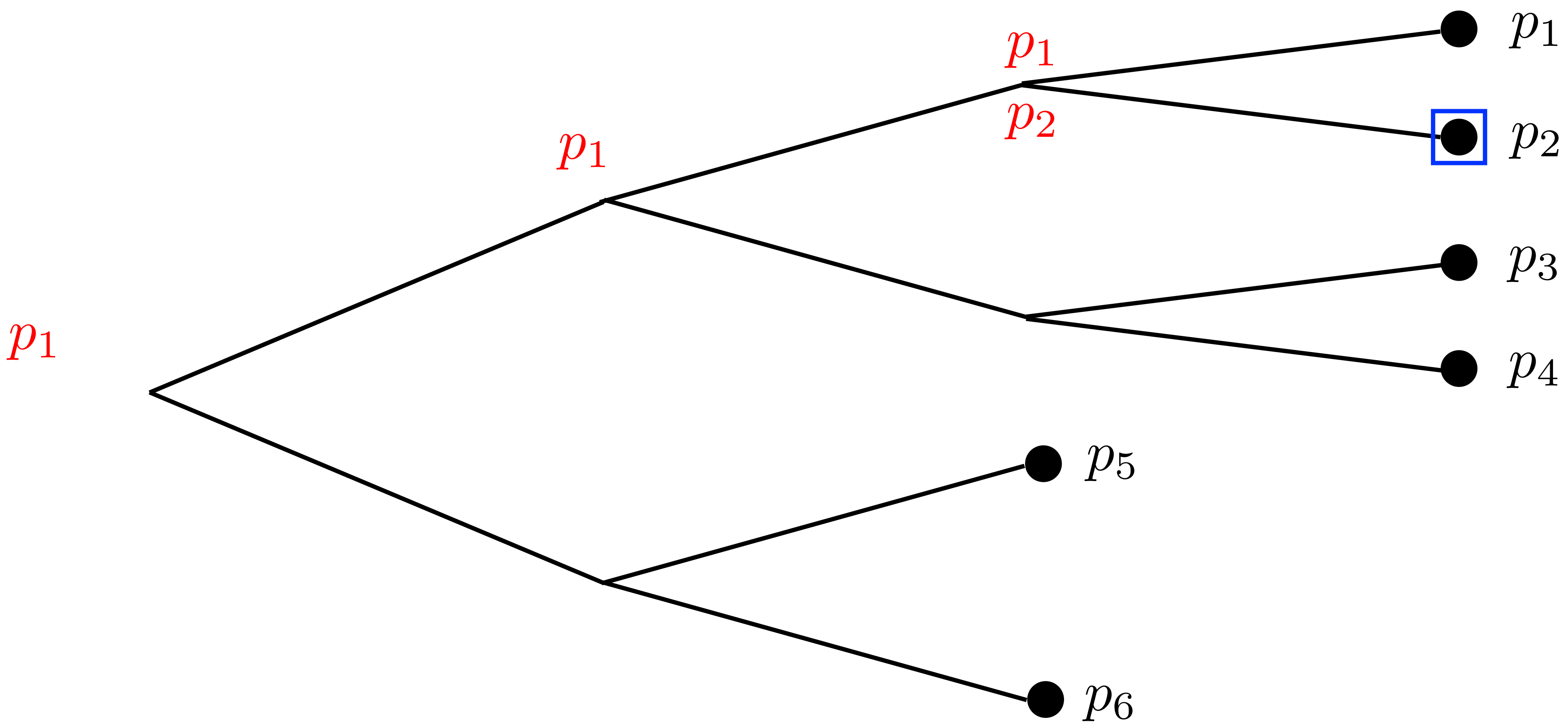
Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

$$\boxed{L = 3p_1 + 3p_2 + 3p_3 + 3p_4 + 2p_5 + 2p_6}$$



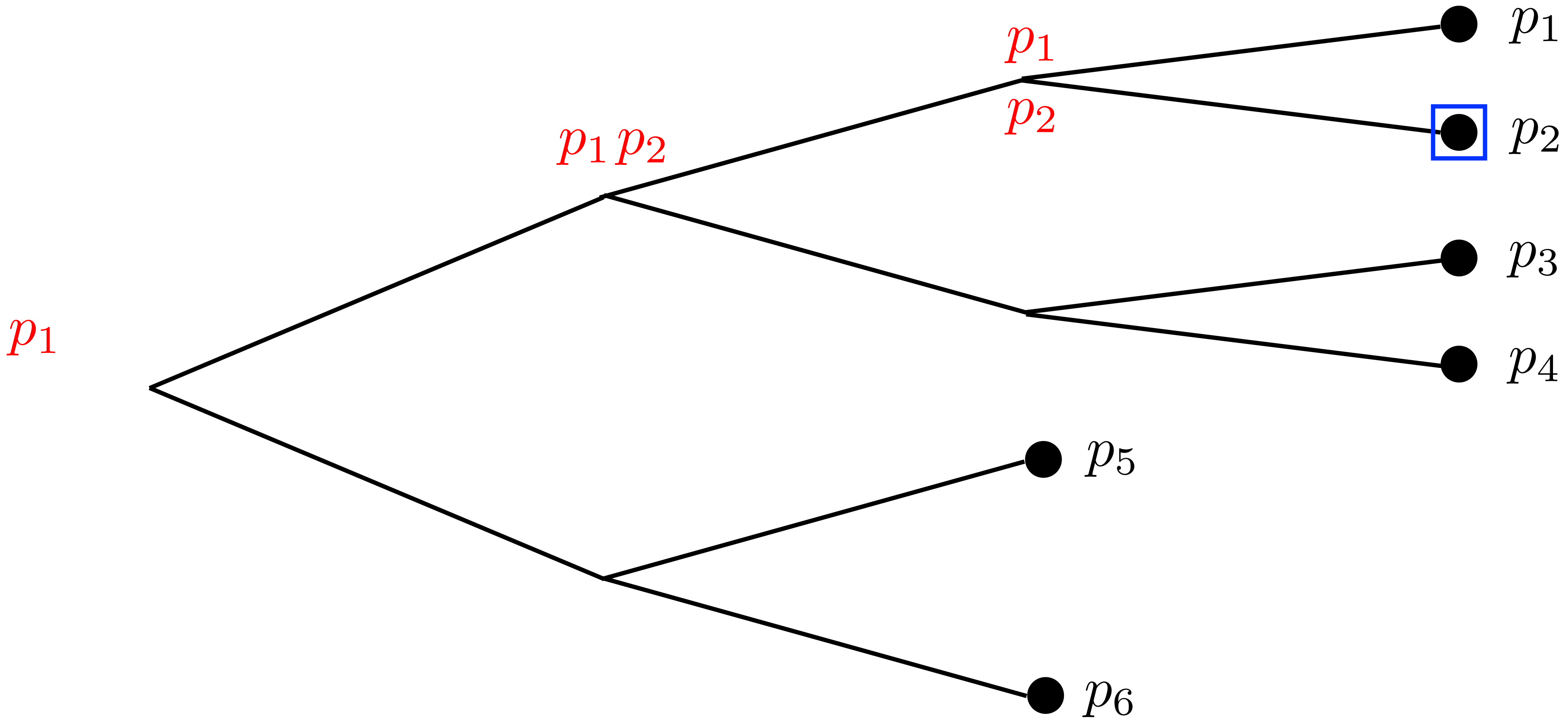
Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

$$\boxed{L = 3p_1 + 3p_2 + 3p_3 + 3p_4 + 2p_5 + 2p_6}$$



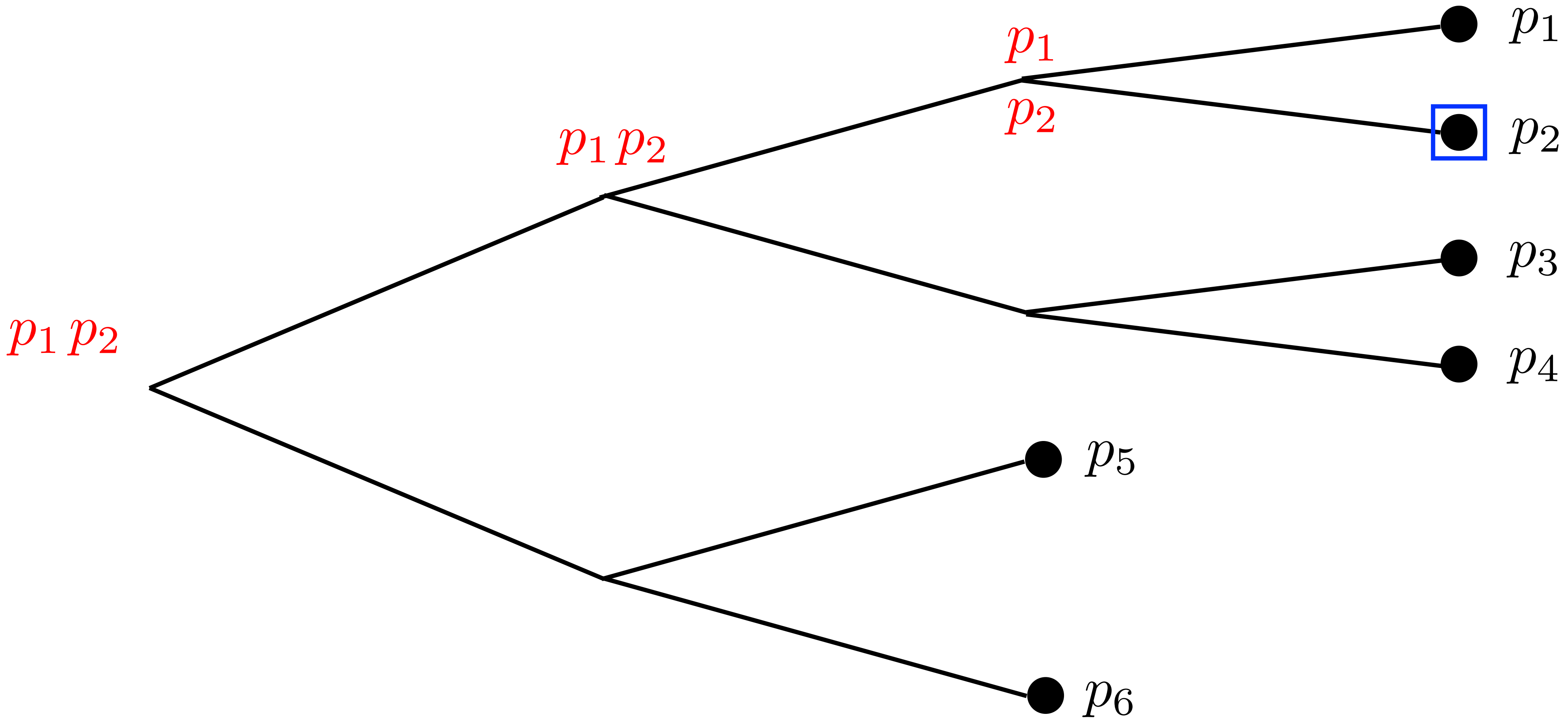
Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

$$\boxed{L = 3p_1 + 3p_2 + 3p_3 + 3p_4 + 2p_5 + 2p_6}$$



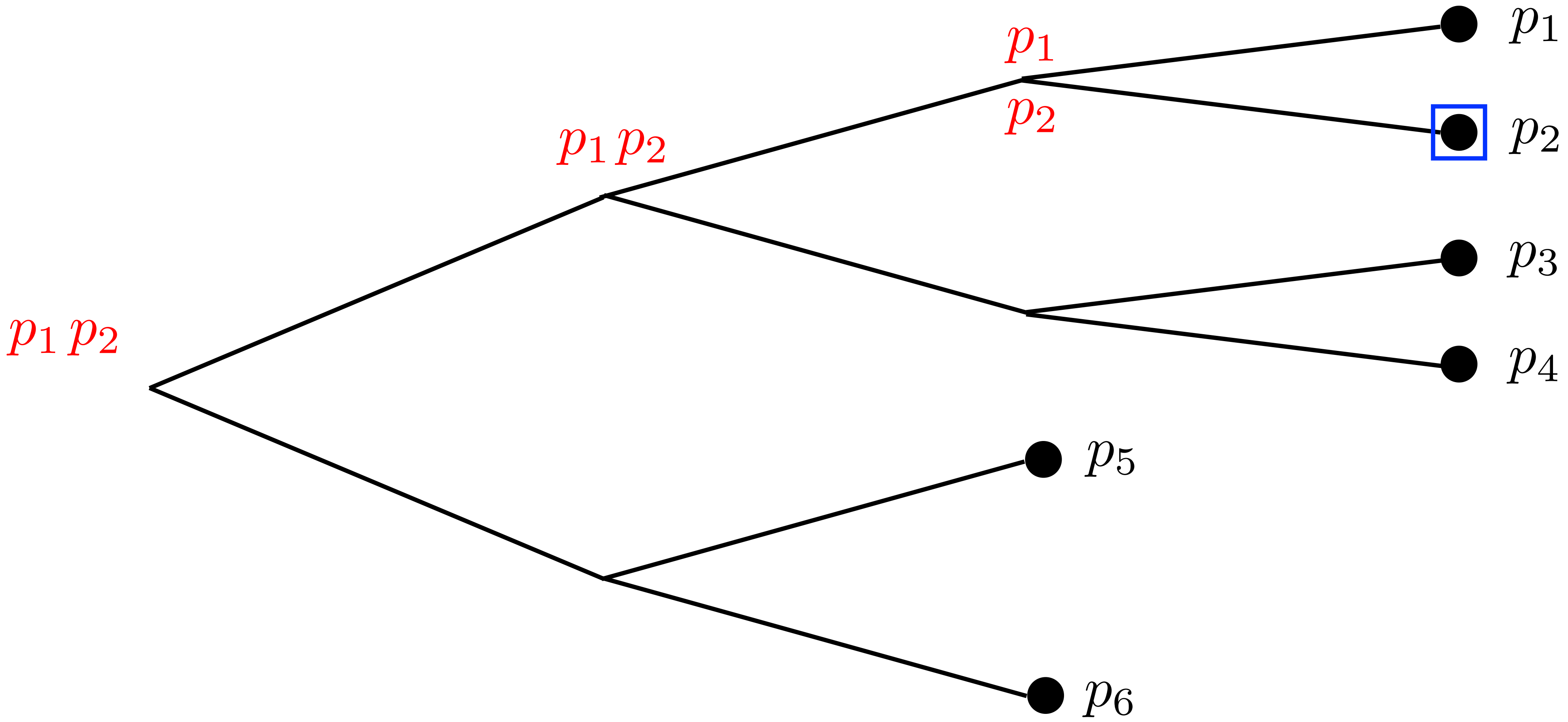
Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

$$\boxed{L = 3p_1 + 3p_2 + 3p_3 + 3p_4 + 2p_5 + 2p_6}$$



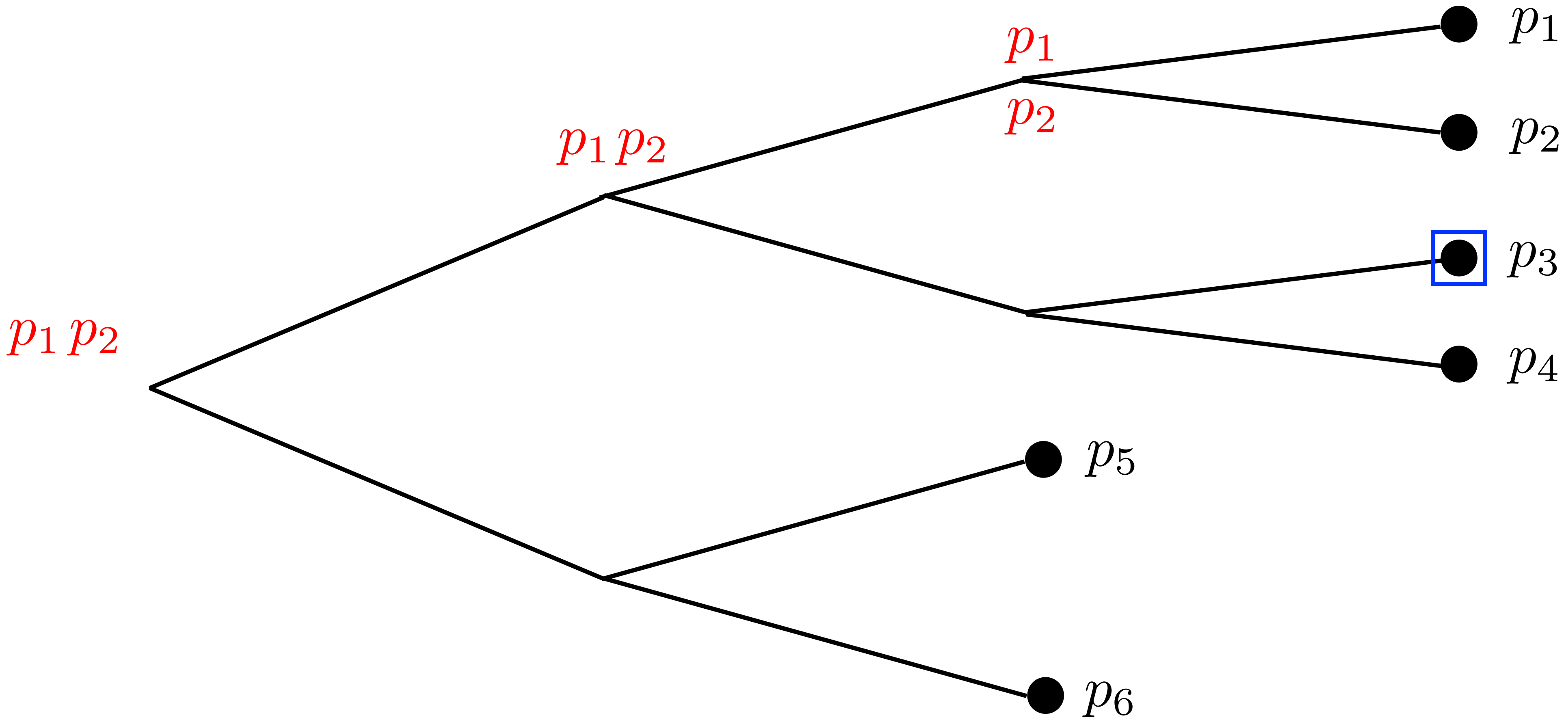
Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

$$\boxed{L = 3p_1 + \underline{3p_2} + 3p_3 + 3p_4 + 2p_5 + 2p_6}$$



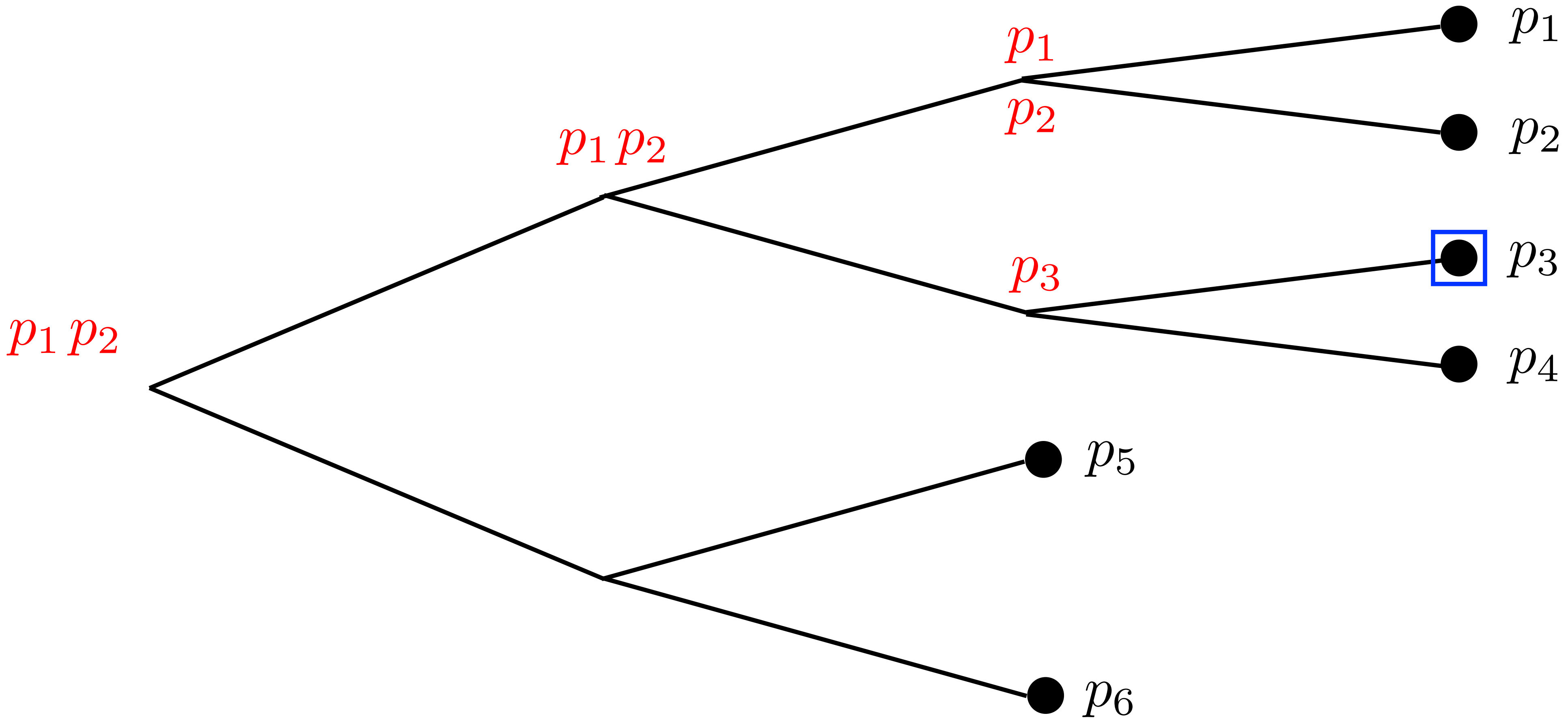
Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

$$\boxed{L = 3p_1 + 3p_2 + 3p_3 + 3p_4 + 2p_5 + 2p_6}$$



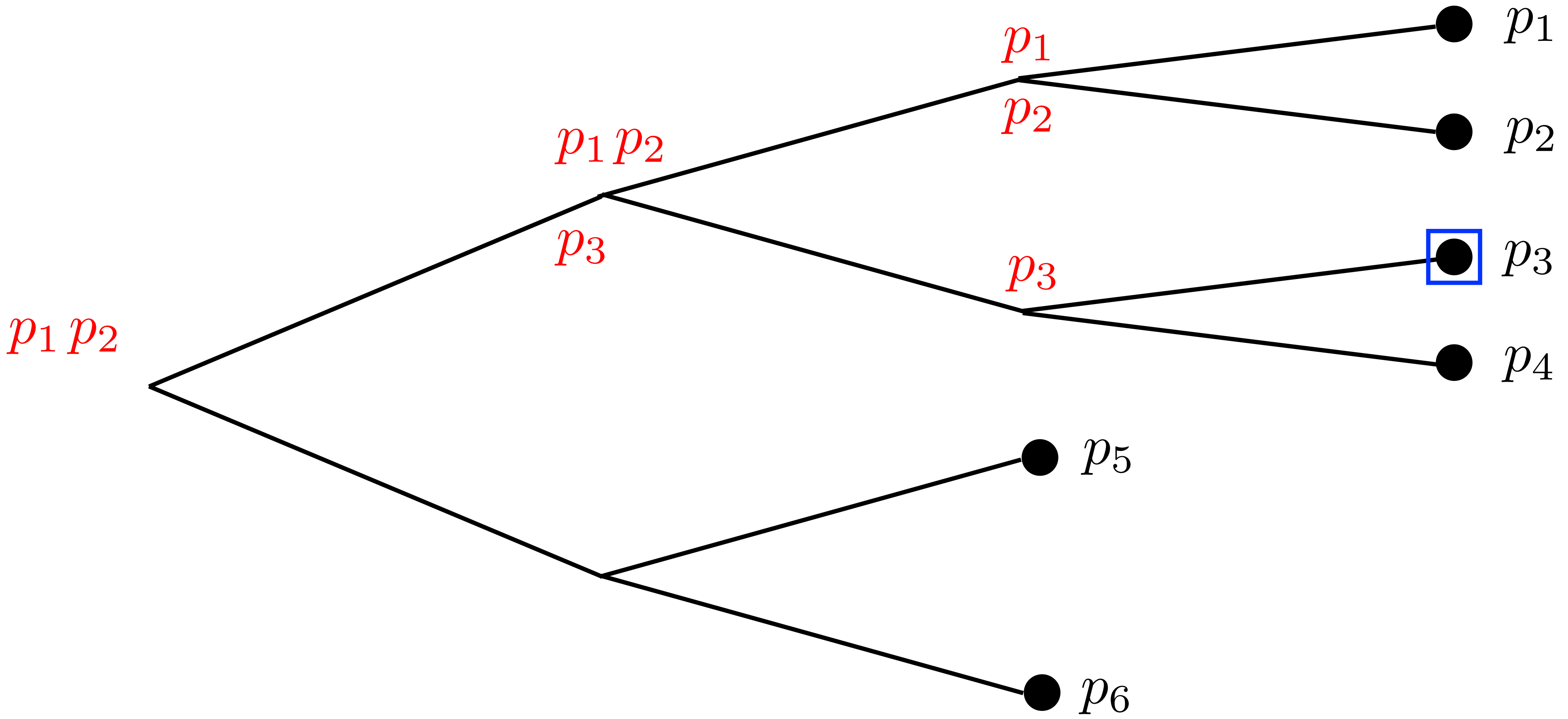
Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

$$\boxed{L = 3p_1 + 3p_2 + 3p_3 + 3p_4 + 2p_5 + 2p_6}$$



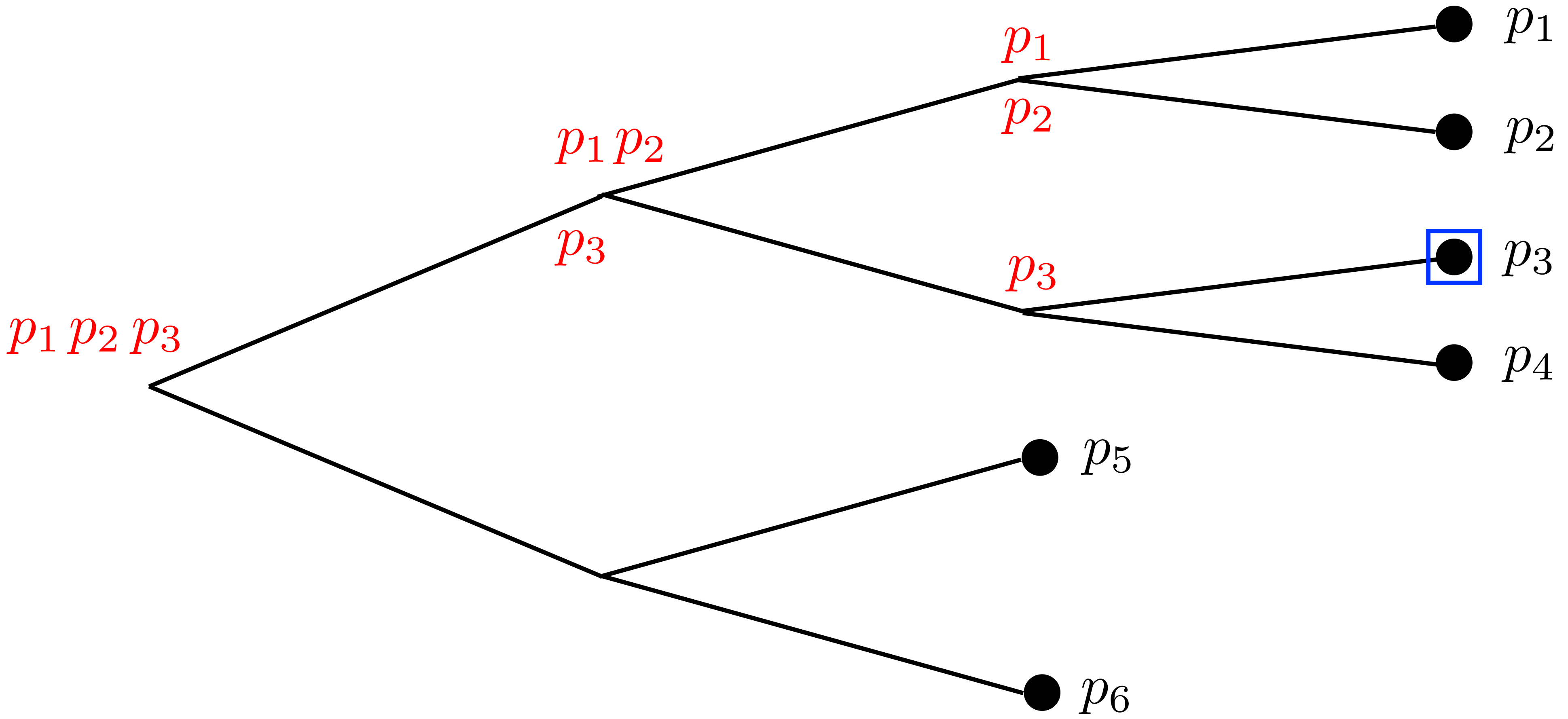
Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

$$\boxed{L = 3p_1 + 3p_2 + 3p_3 + 3p_4 + 2p_5 + 2p_6}$$



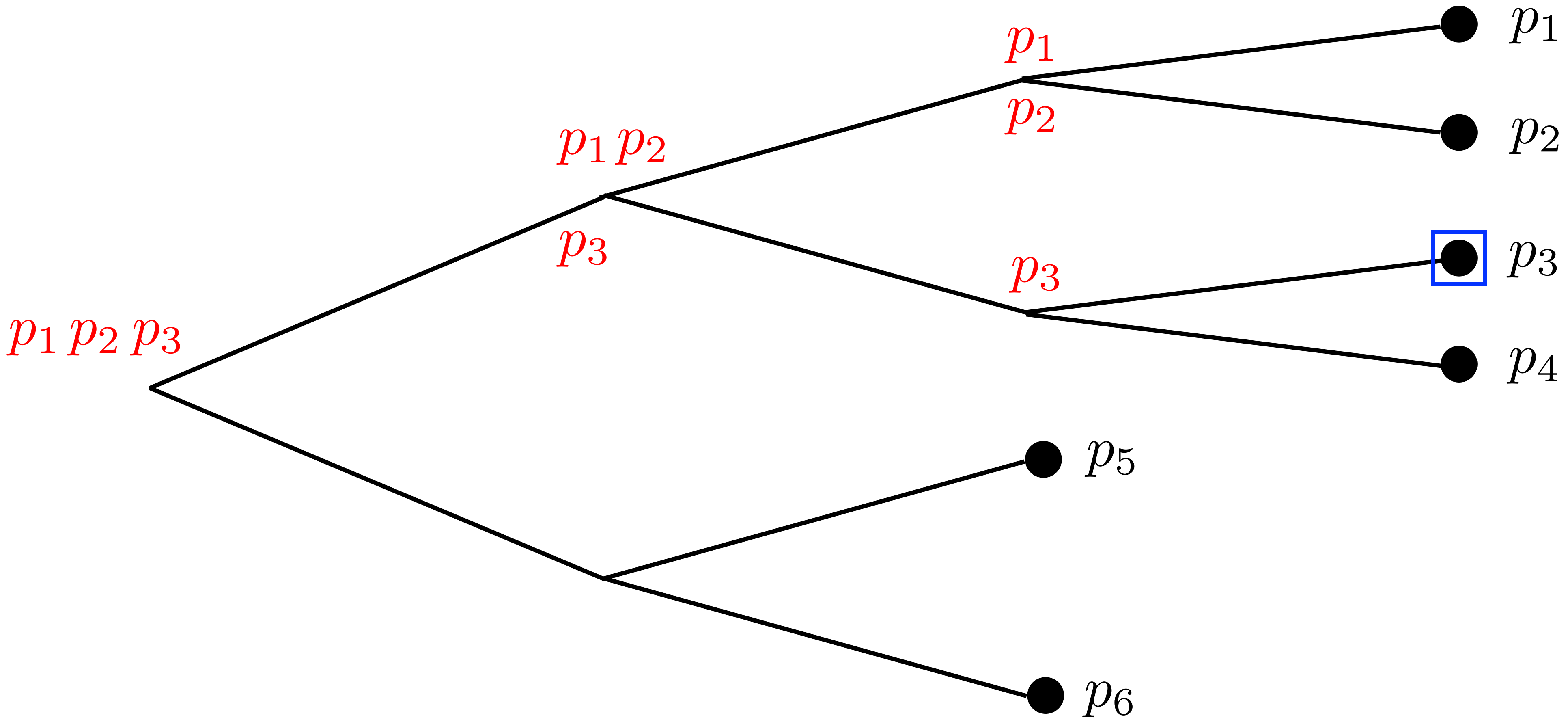
Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

$$\boxed{L = 3p_1 + 3p_2 + 3p_3 + 3p_4 + 2p_5 + 2p_6}$$



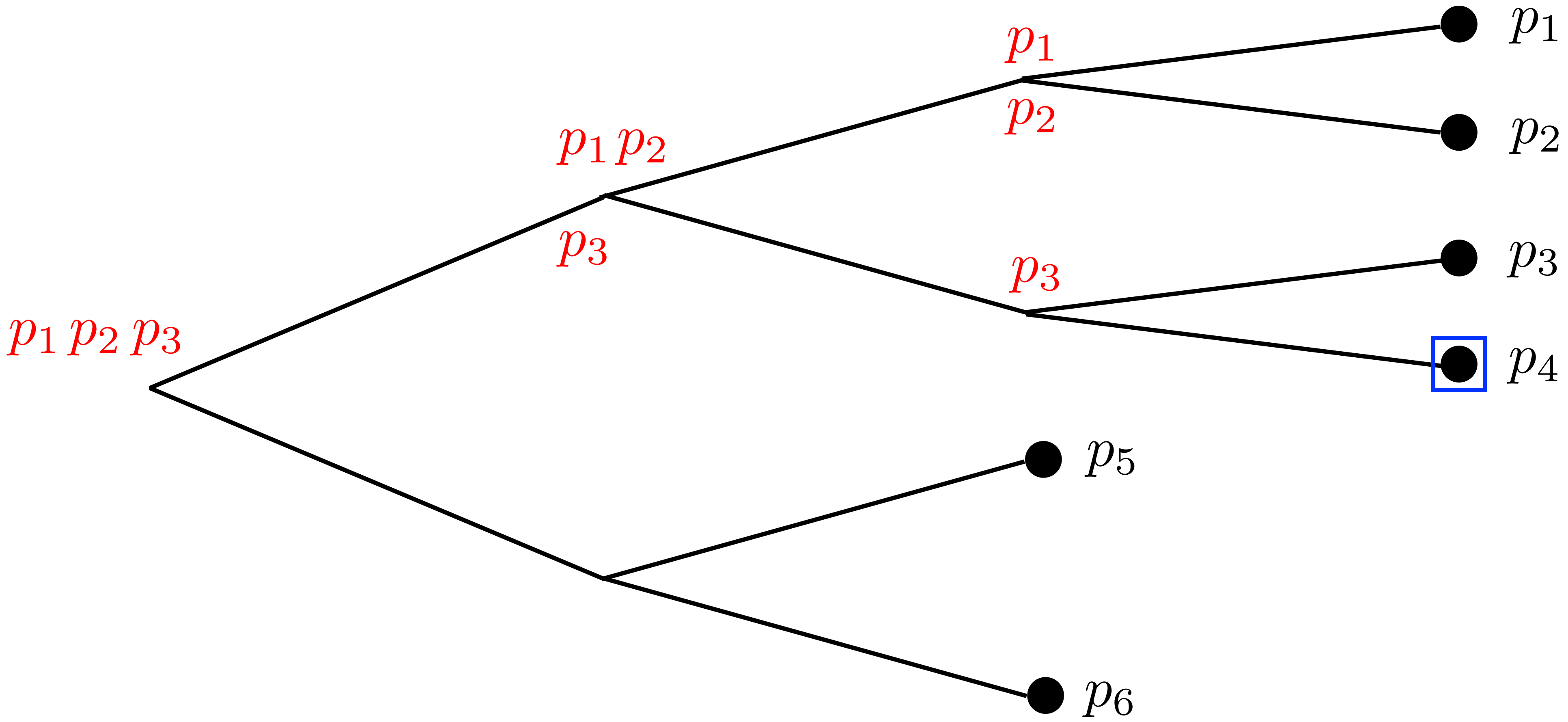
Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

$$\boxed{L = 3p_1 + 3p_2 + \underline{3p_3} + 3p_4 + 2p_5 + 2p_6}$$



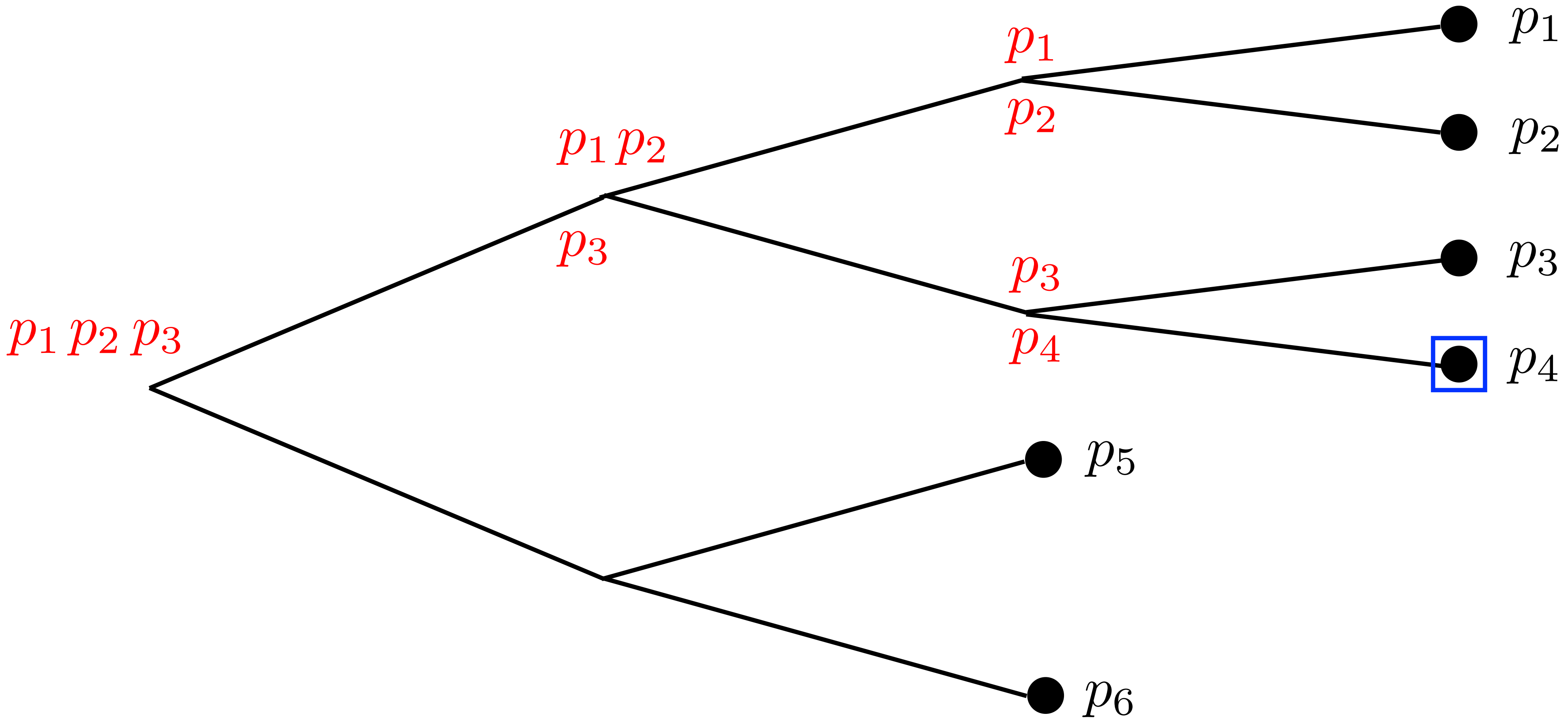
Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

$$\boxed{L = 3p_1 + 3p_2 + 3p_3 + 3p_4 + 2p_5 + 2p_6}$$



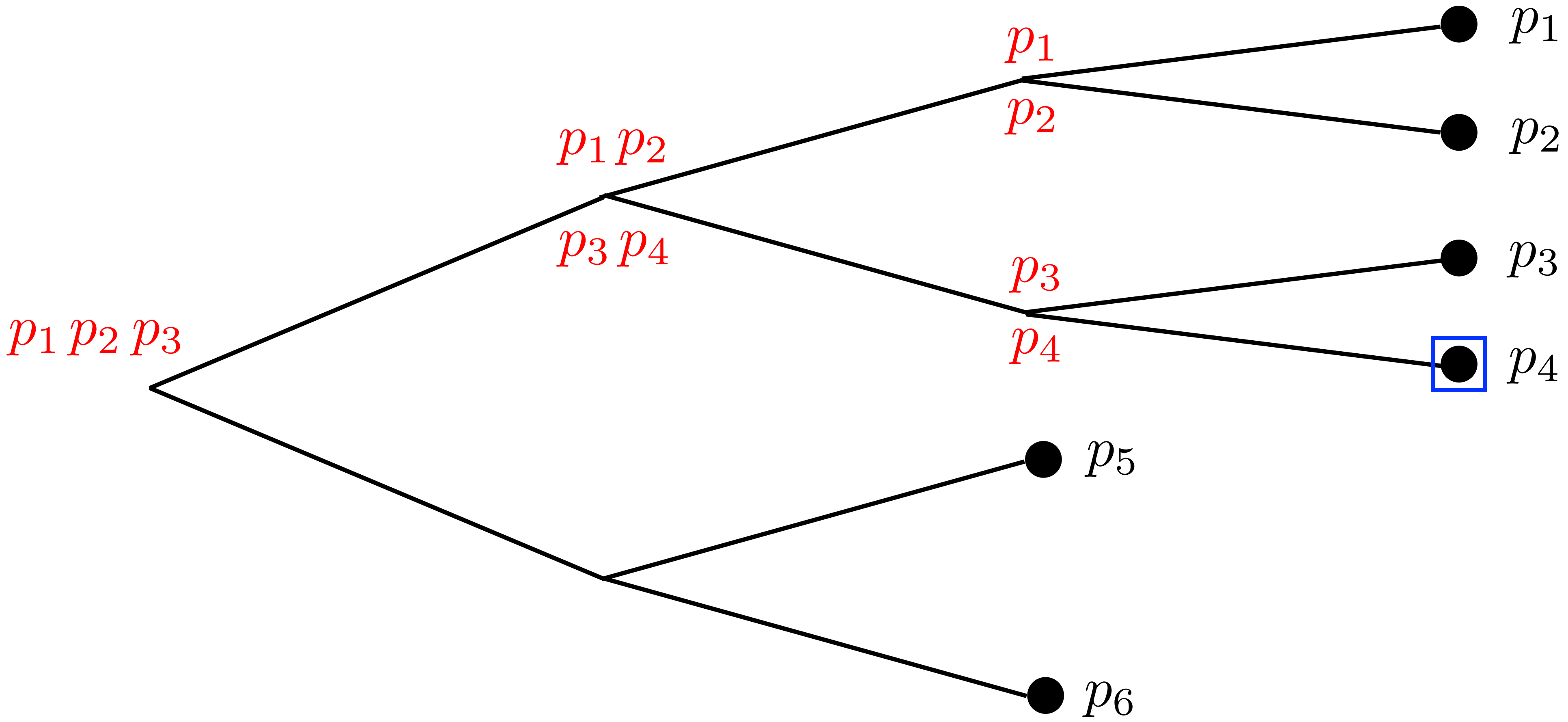
Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

$$\boxed{L = 3p_1 + 3p_2 + 3p_3 + 3p_4 + 2p_5 + 2p_6}$$



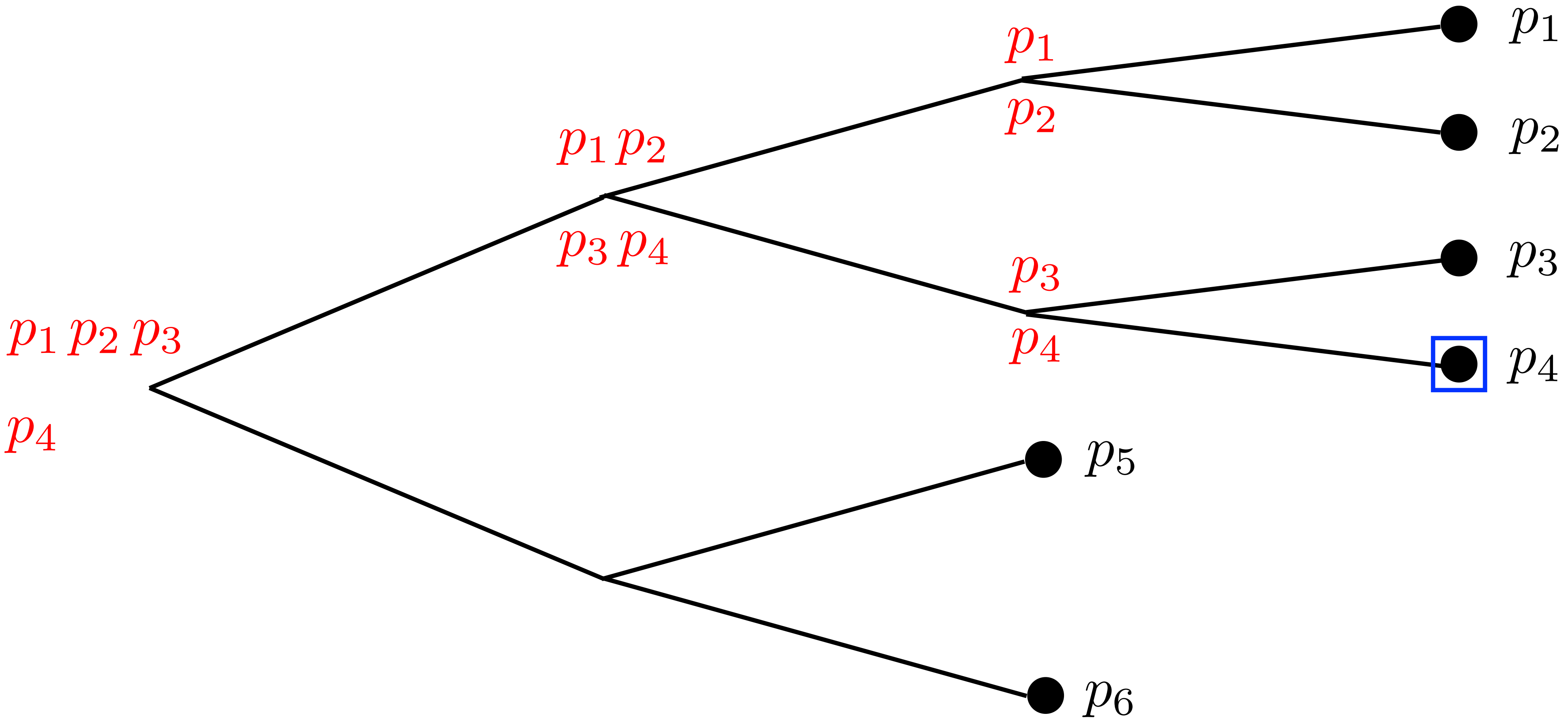
Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

$$\boxed{L = 3p_1 + 3p_2 + 3p_3 + 3p_4 + 2p_5 + 2p_6}$$



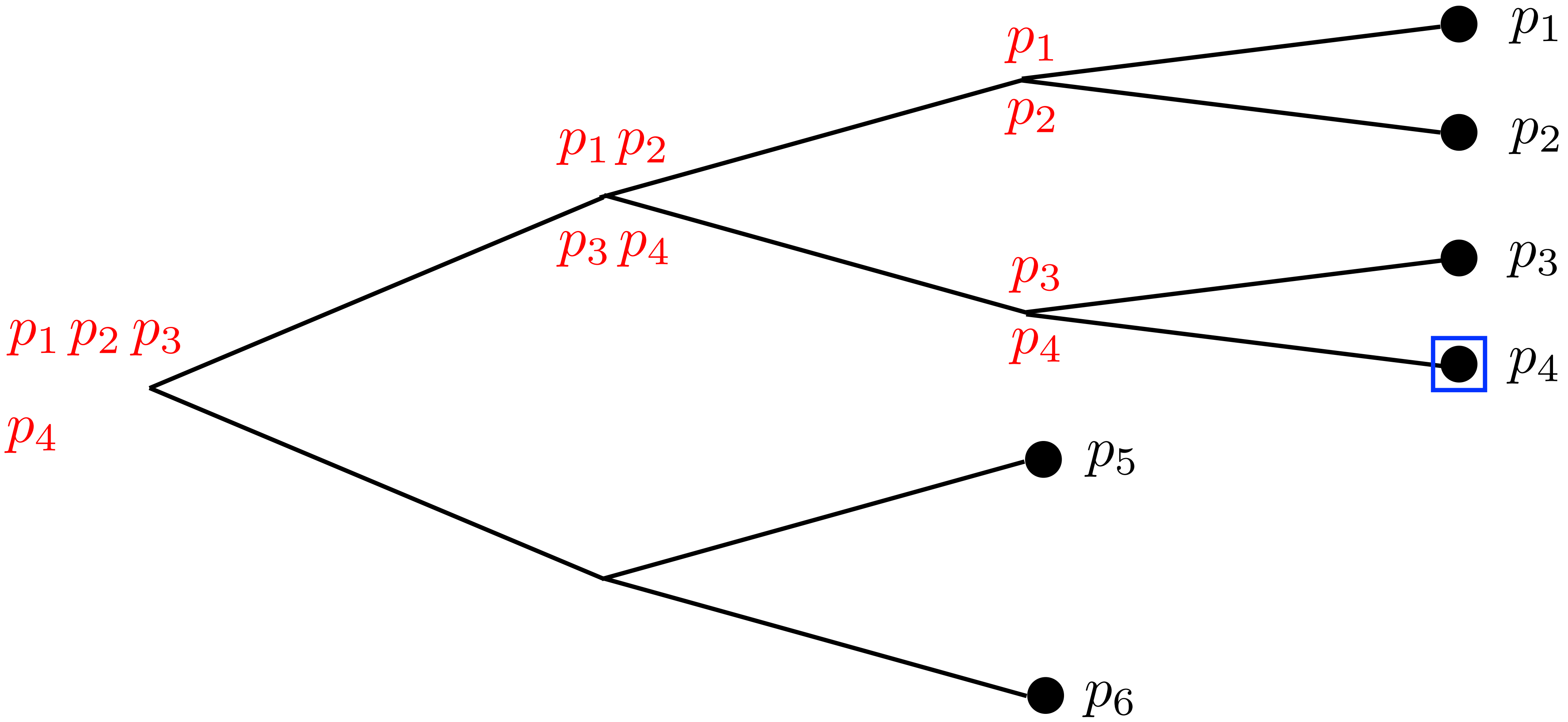
Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

$$\boxed{L = 3p_1 + 3p_2 + 3p_3 + 3p_4 + 2p_5 + 2p_6}$$



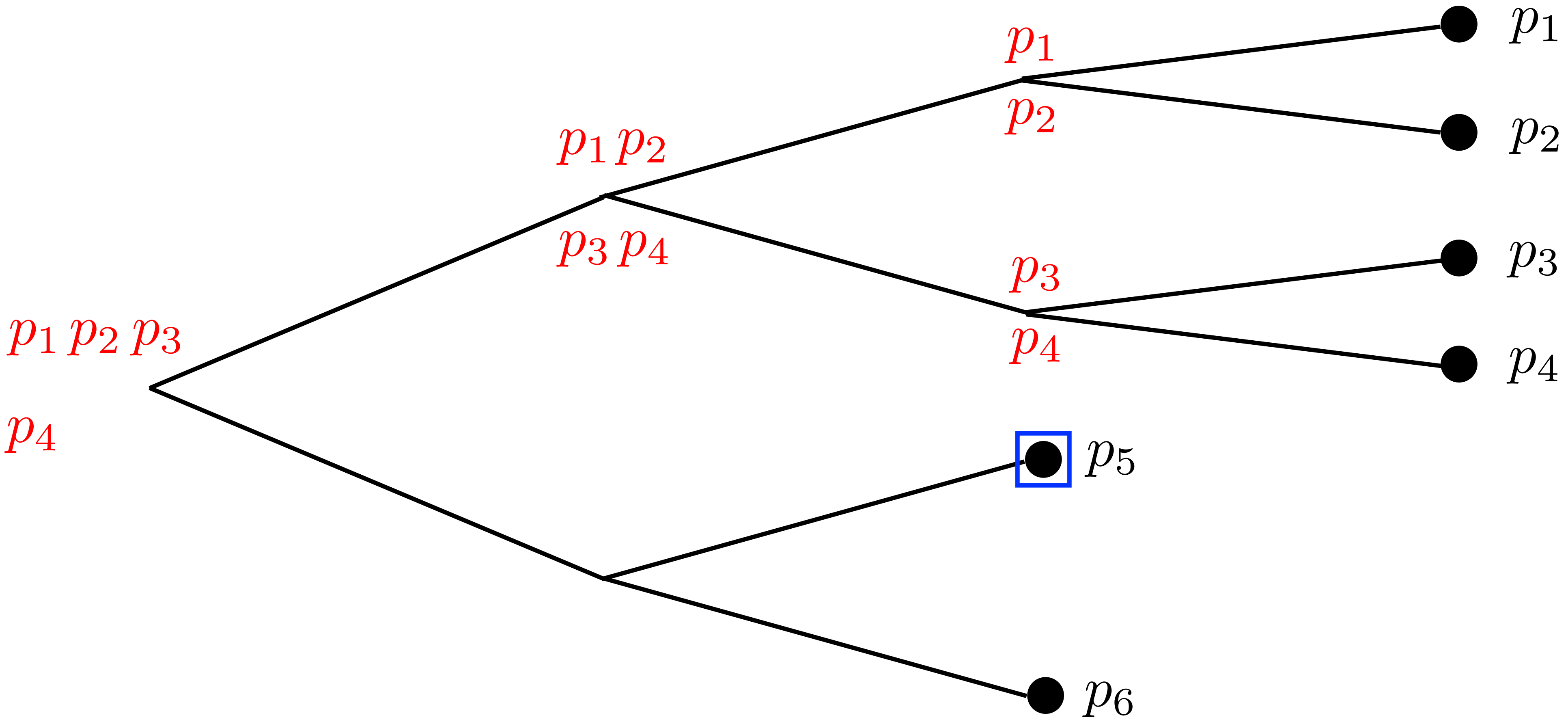
Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

$$\boxed{L = 3p_1 + 3p_2 + 3p_3 + \underline{3p_4} + 2p_5 + 2p_6}$$



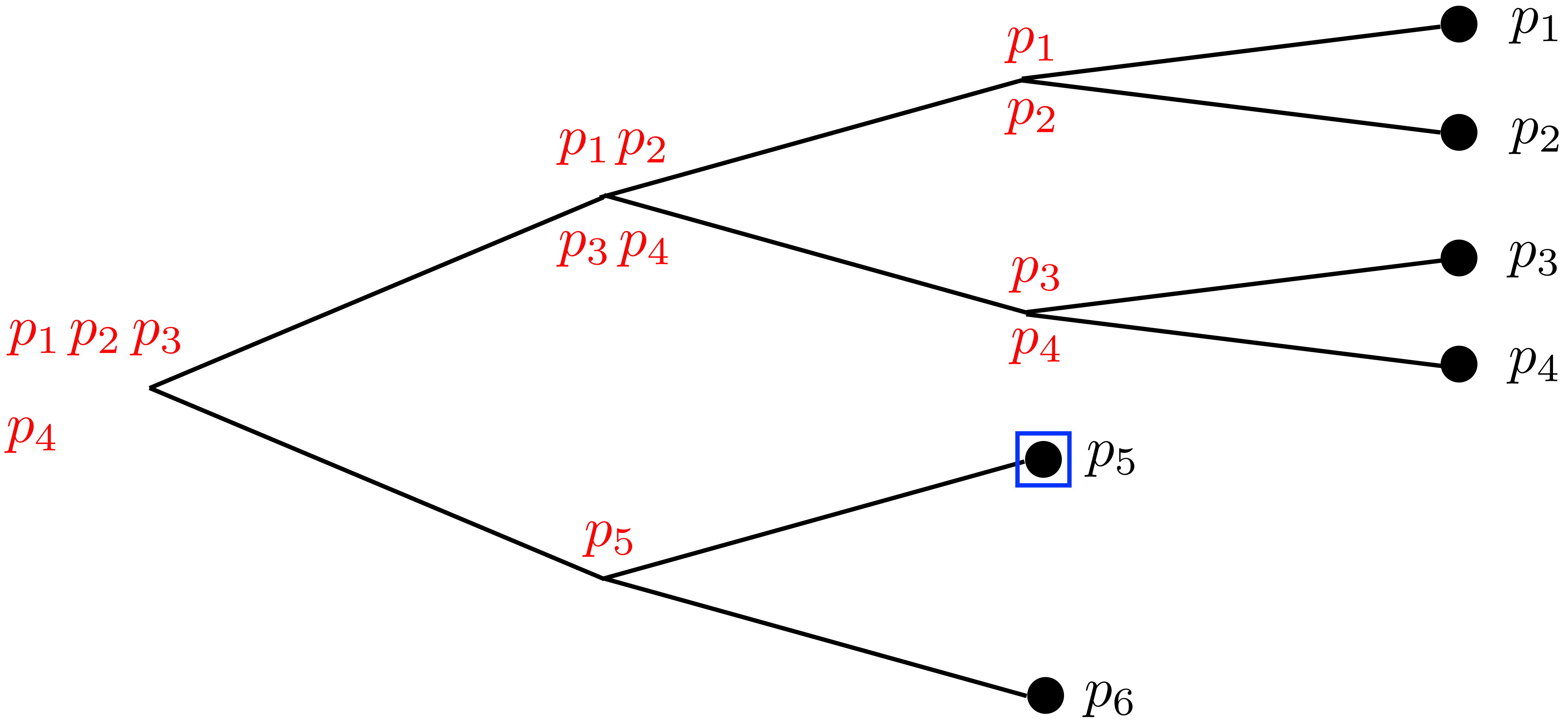
Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

$$\boxed{L = 3p_1 + 3p_2 + 3p_3 + 3p_4 + 2p_5 + 2p_6}$$



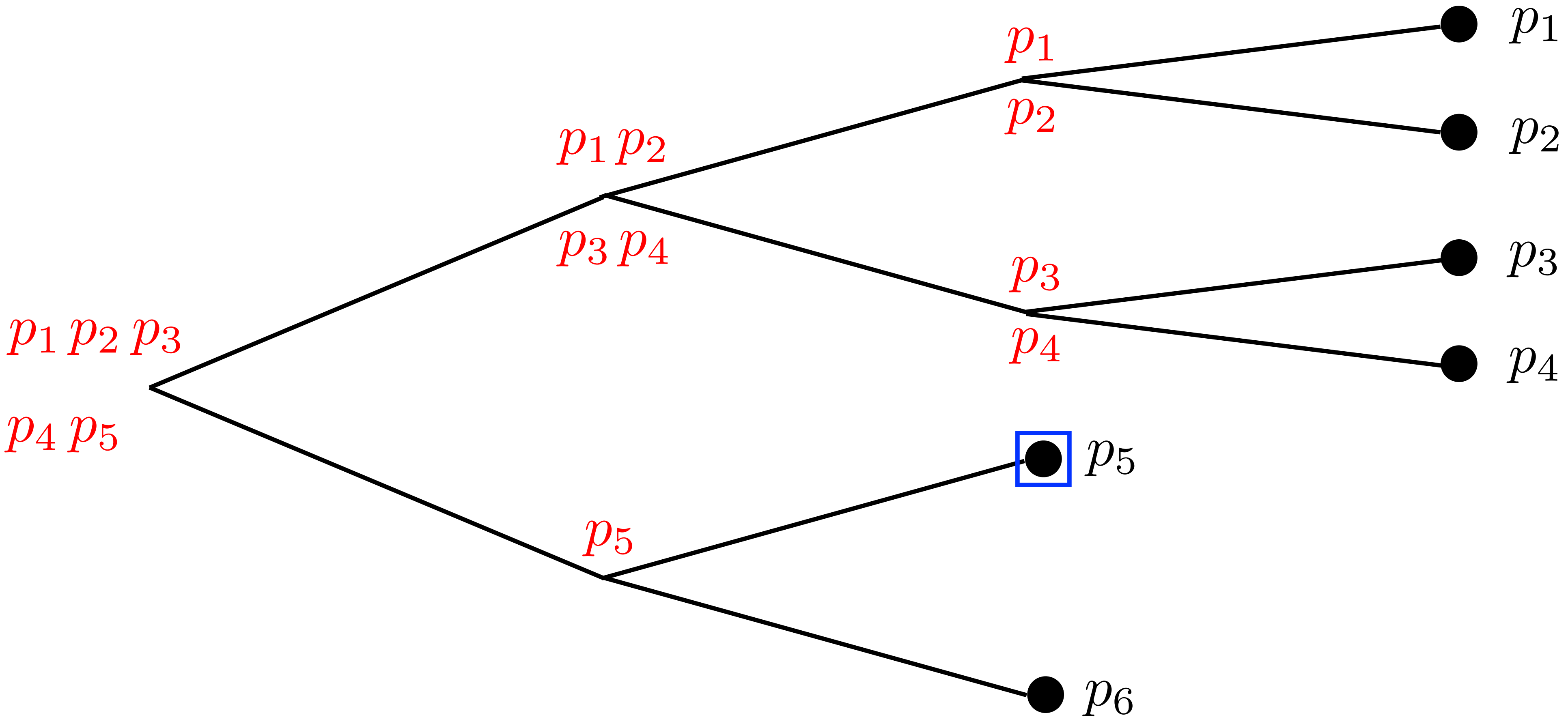
Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

$$\boxed{L = 3p_1 + 3p_2 + 3p_3 + 3p_4 + 2p_5 + 2p_6}$$



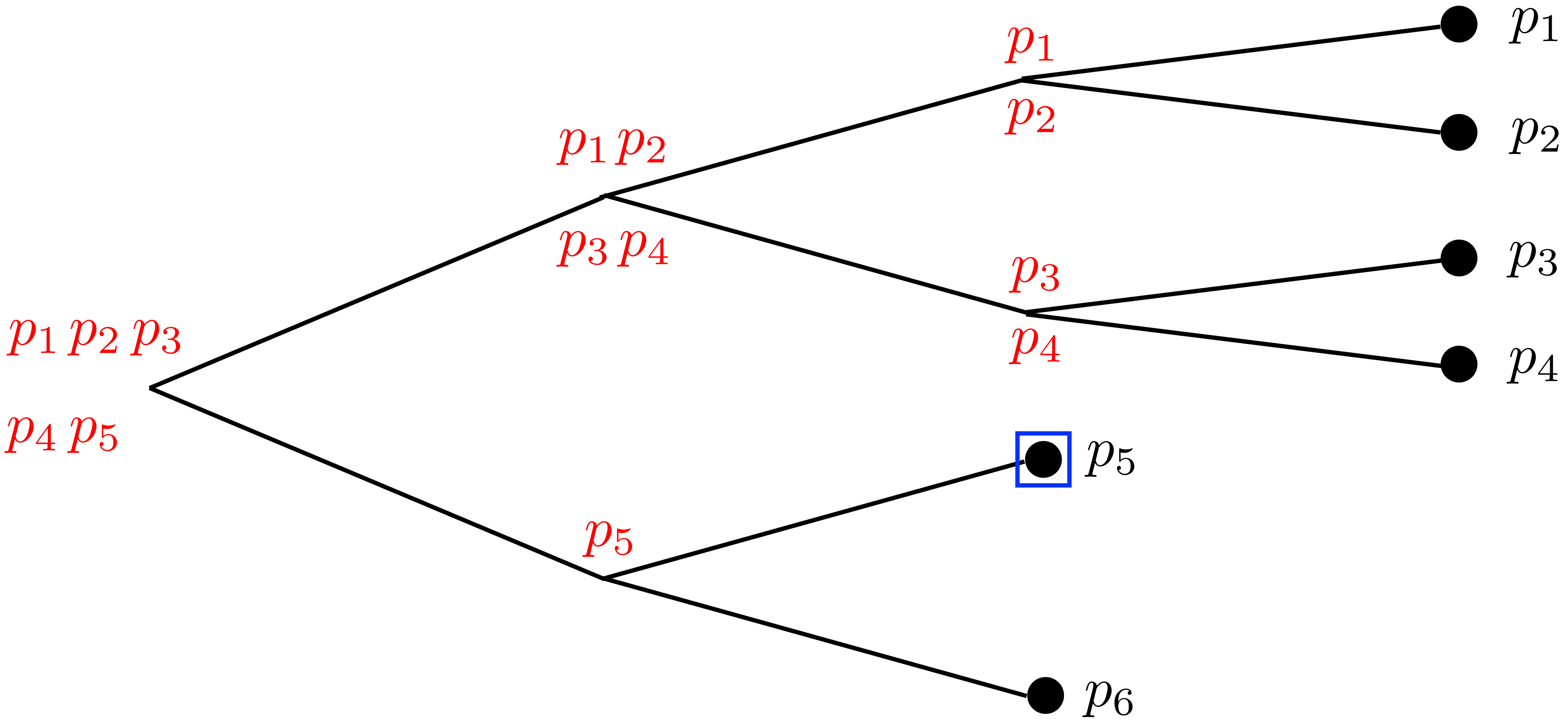
Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

$$\boxed{L = 3p_1 + 3p_2 + 3p_3 + 3p_4 + 2p_5 + 2p_6}$$



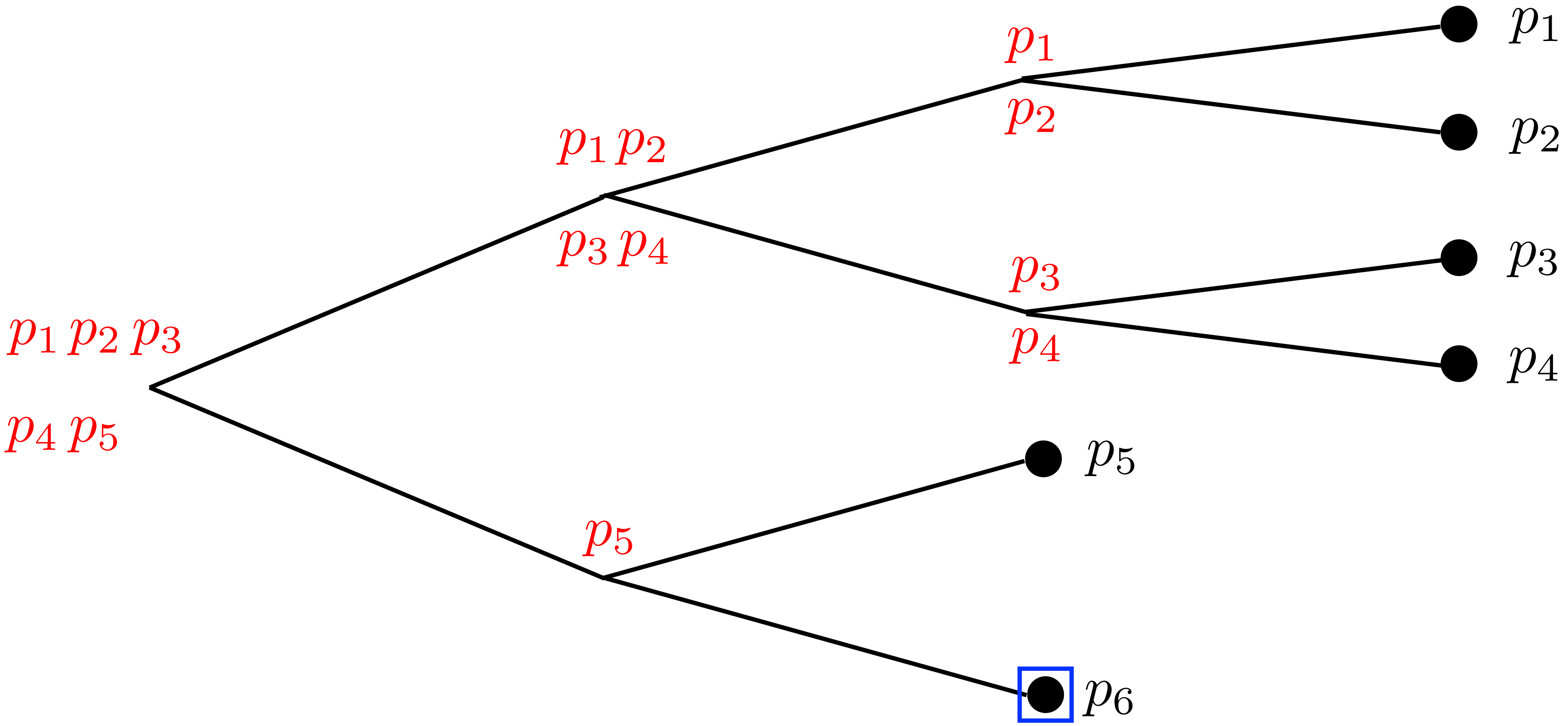
Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

$$\boxed{L = 3p_1 + 3p_2 + 3p_3 + 3p_4 + \underline{2p_5} + 2p_6}$$



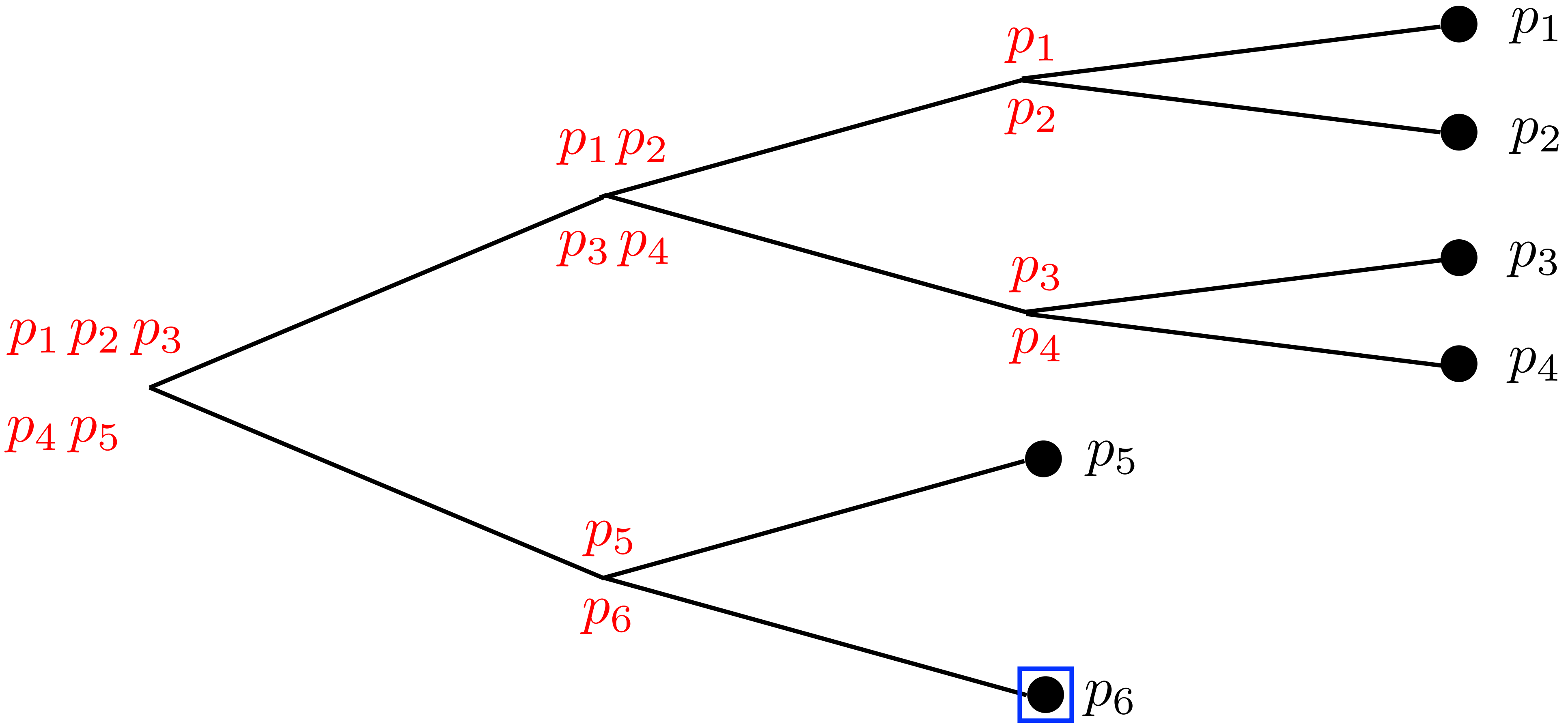
Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

$$\boxed{L = 3p_1 + 3p_2 + 3p_3 + 3p_4 + 2p_5 + 2p_6}$$



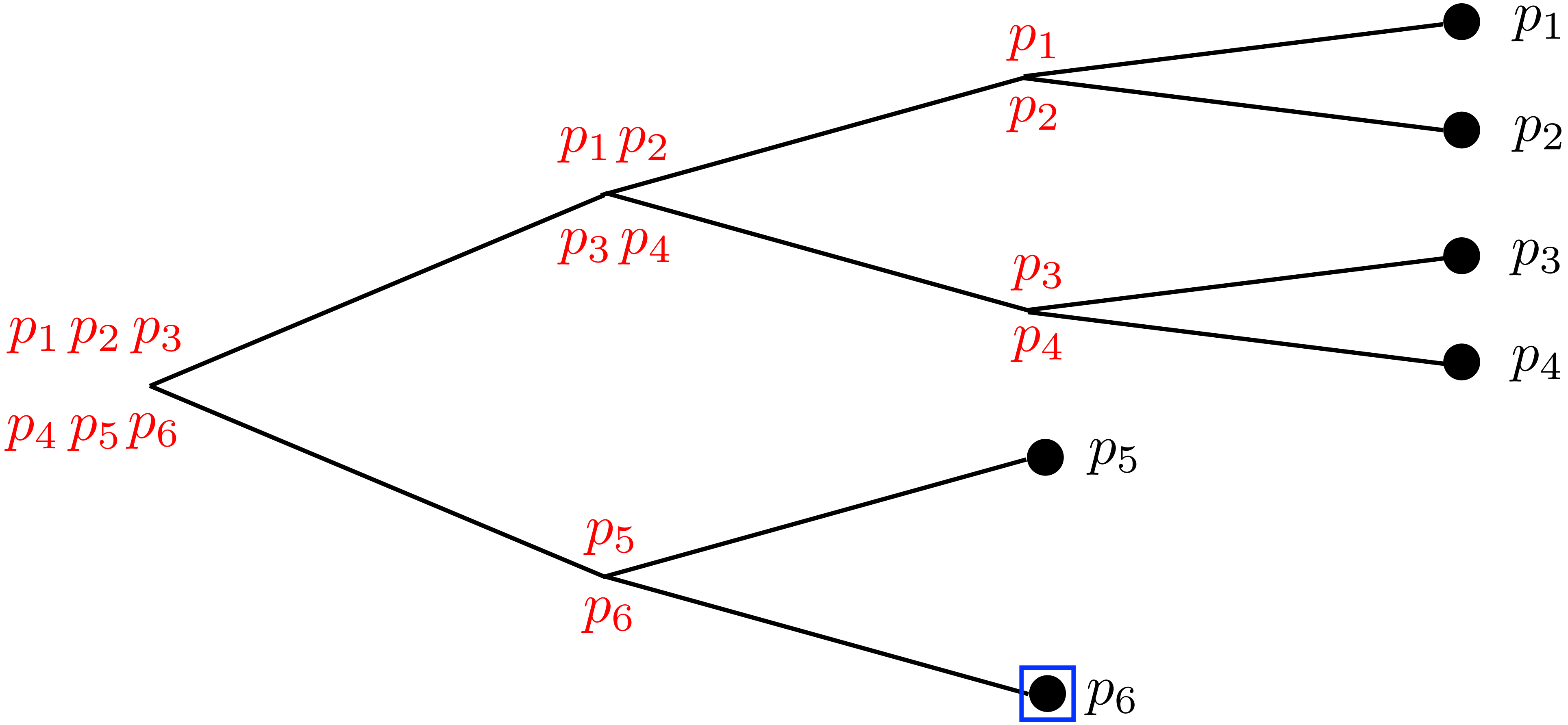
Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

$$\boxed{L = 3p_1 + 3p_2 + 3p_3 + 3p_4 + 2p_5 + 2p_6}$$



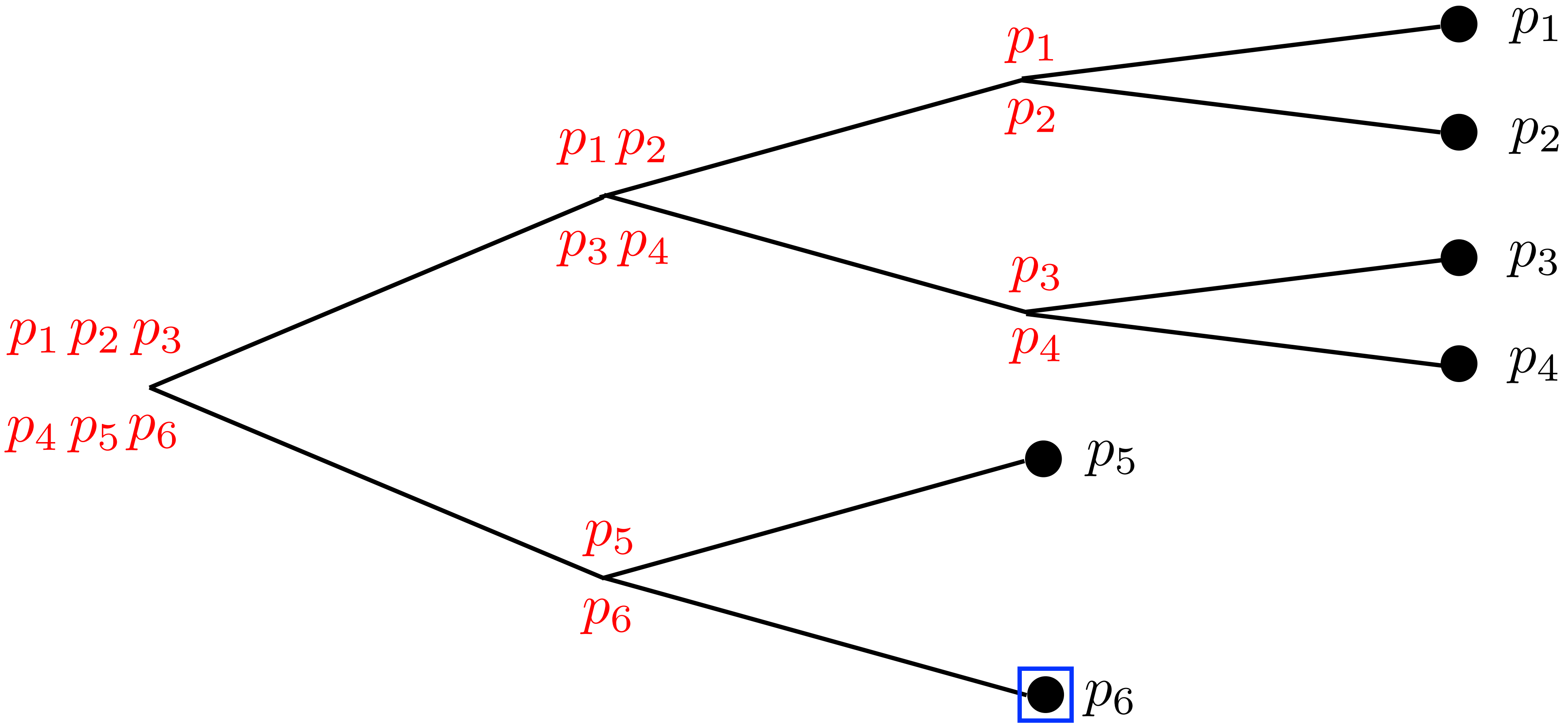
Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

$$\boxed{L = 3p_1 + 3p_2 + 3p_3 + 3p_4 + 2p_5 + 2p_6}$$



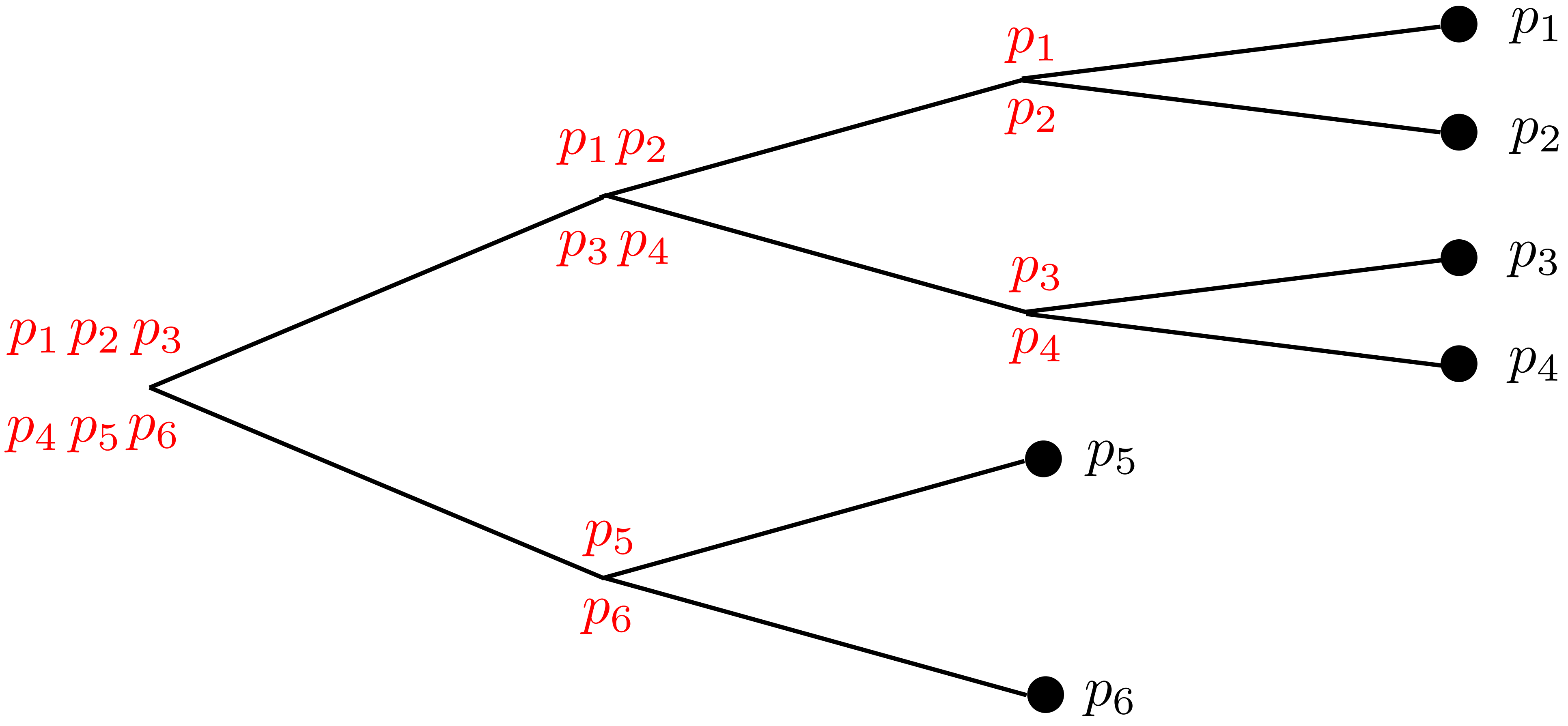
Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

$$\boxed{L = 3p_1 + 3p_2 + 3p_3 + 3p_4 + 2p_5 + \underline{2p_6}}$$



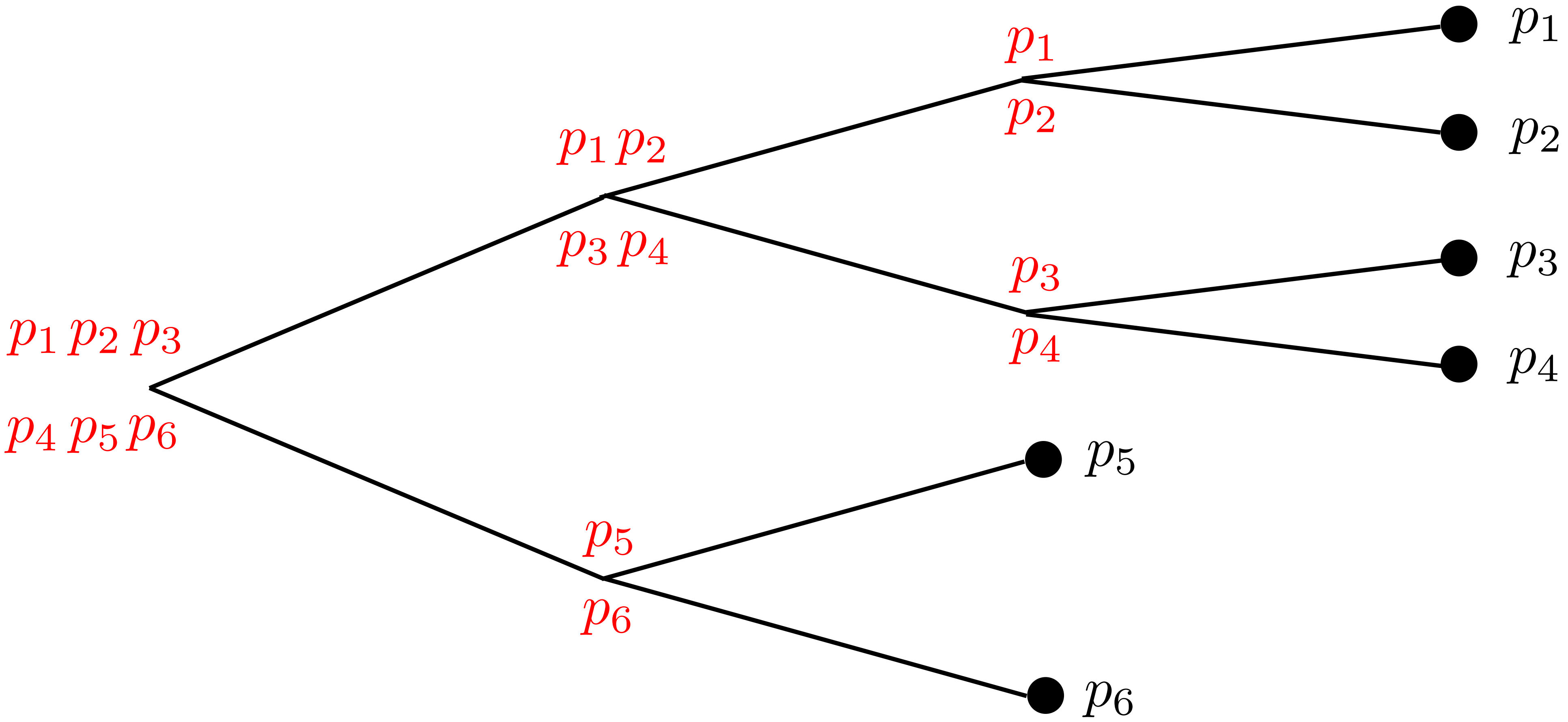
Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

$$\boxed{L = 3p_1 + 3p_2 + 3p_3 + 3p_4 + 2p_5 + 2p_6}$$



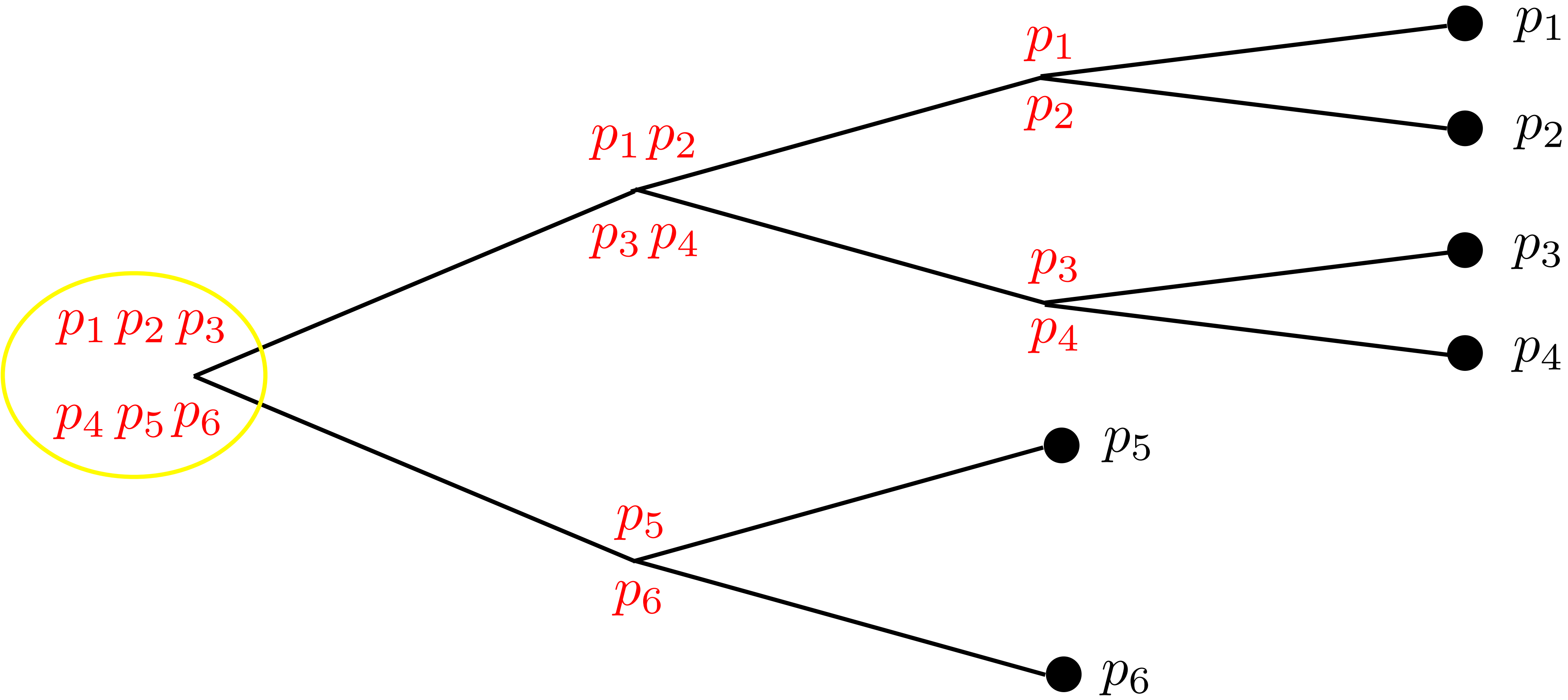
Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

$$\boxed{L = 3p_1 + 3p_2 + 3p_3 + 3p_4 + 2p_5 + 2p_6}$$



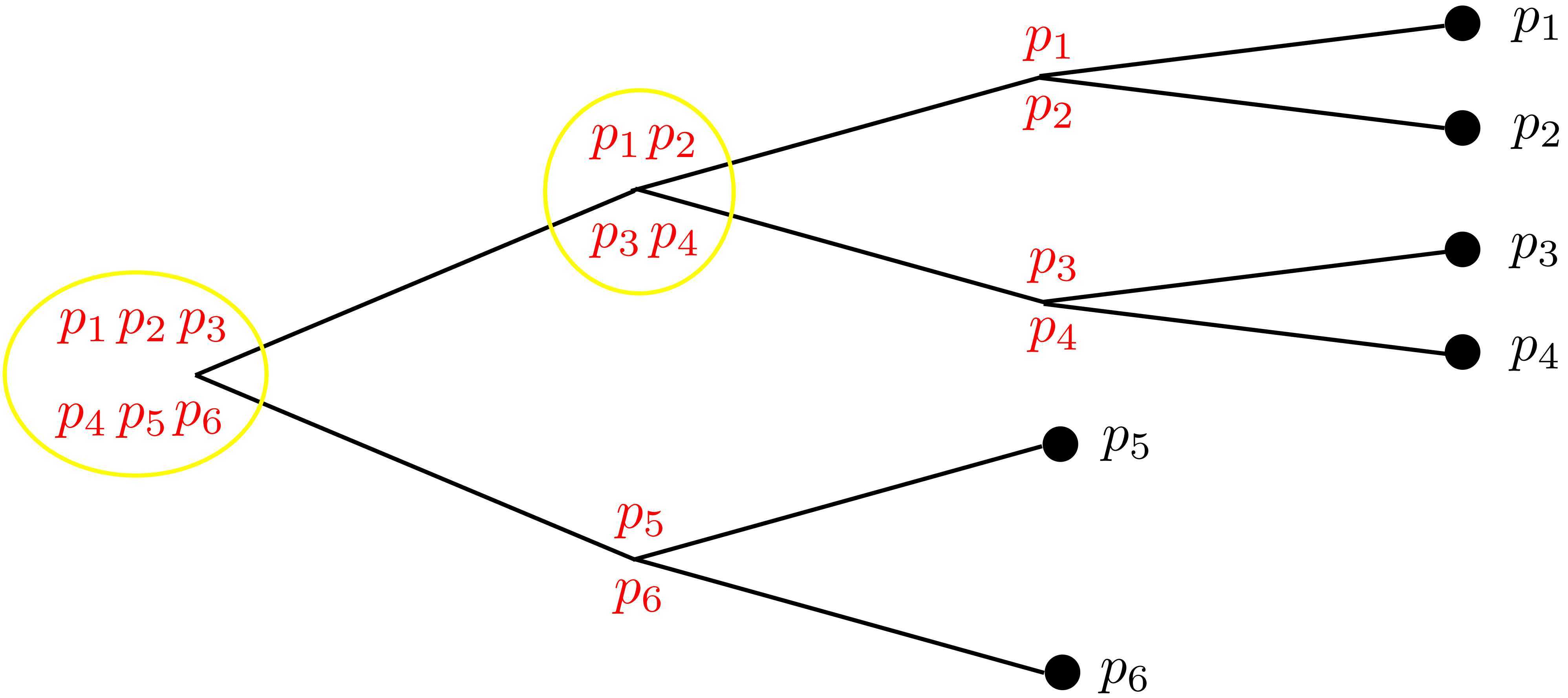
Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

$$\boxed{L = 3p_1 + 3p_2 + 3p_3 + 3p_4 + 2p_5 + 2p_6}$$



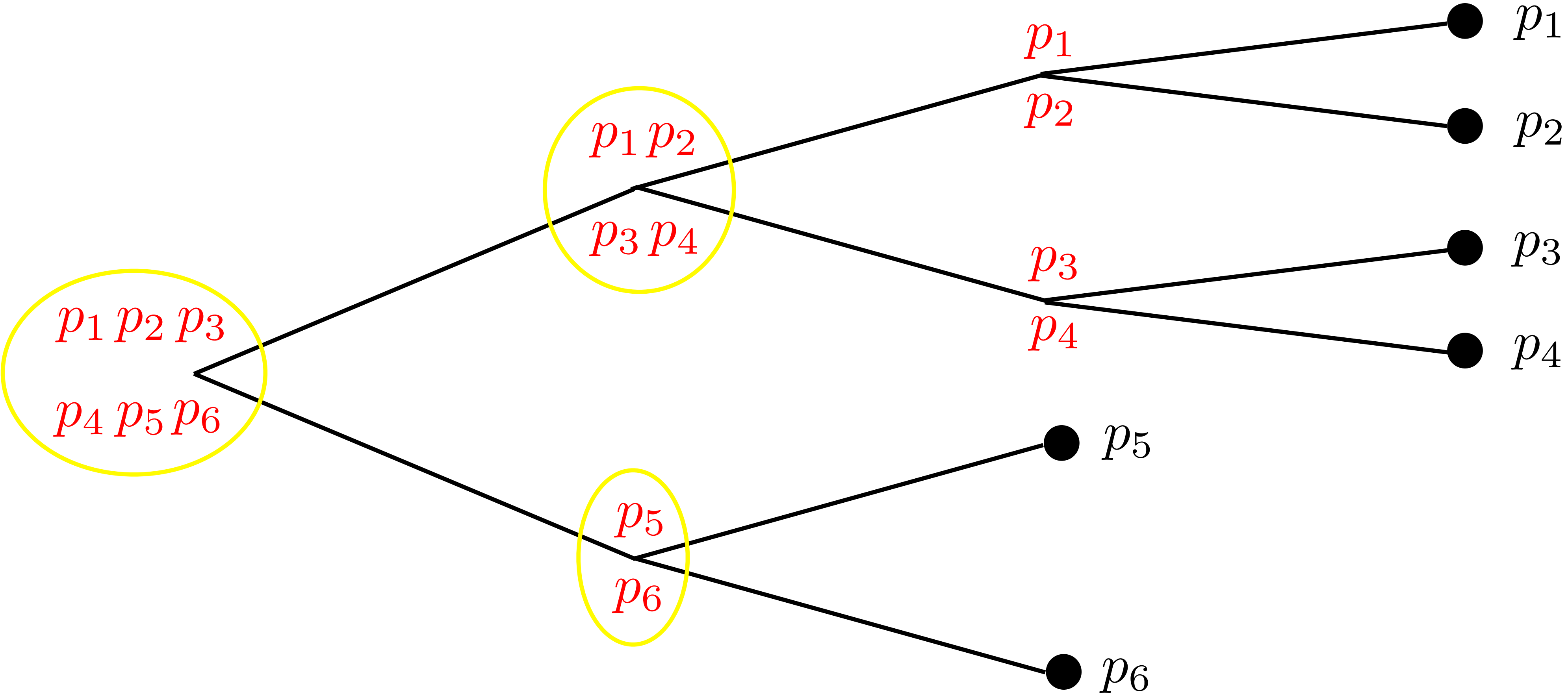
Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

$$\boxed{L = 3p_1 + 3p_2 + 3p_3 + 3p_4 + 2p_5 + 2p_6}$$



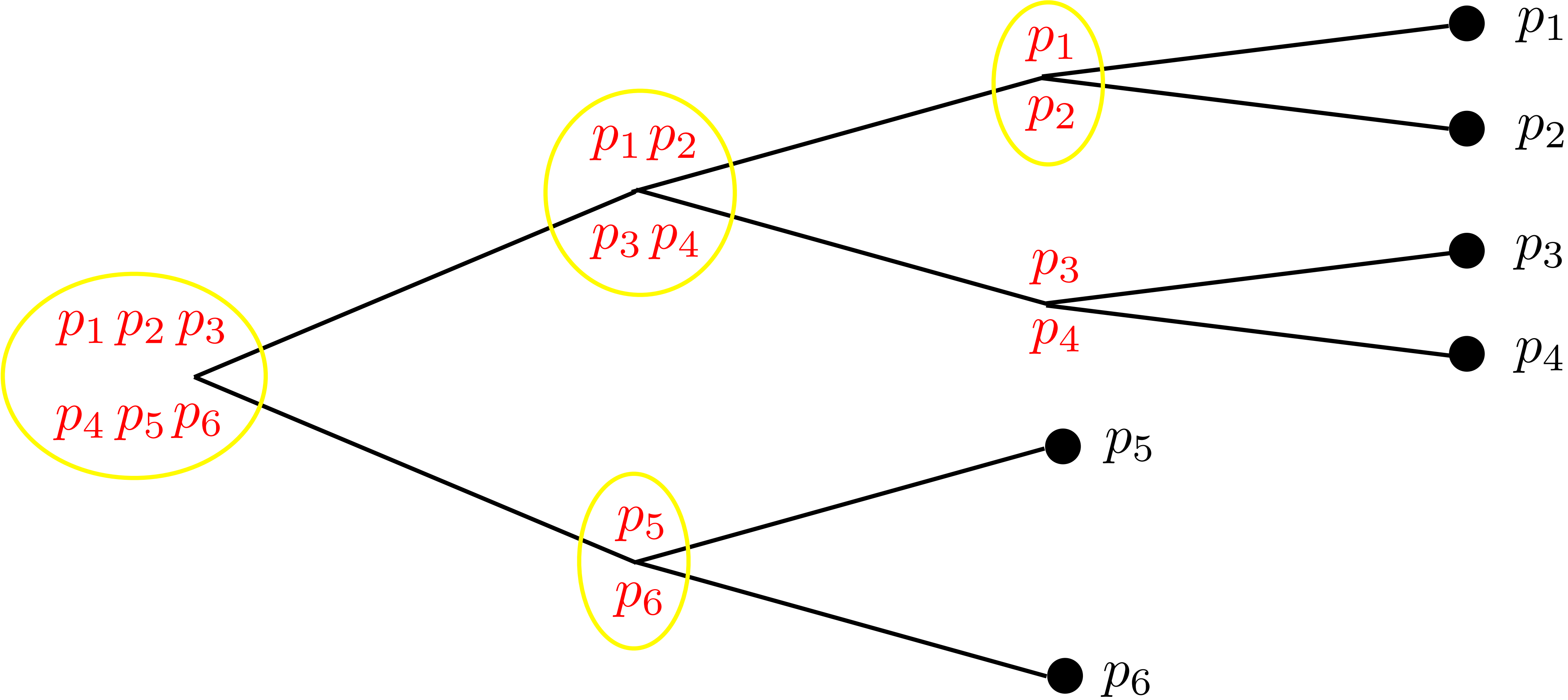
Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

$$\boxed{L = 3p_1 + 3p_2 + 3p_3 + 3p_4 + 2p_5 + 2p_6}$$



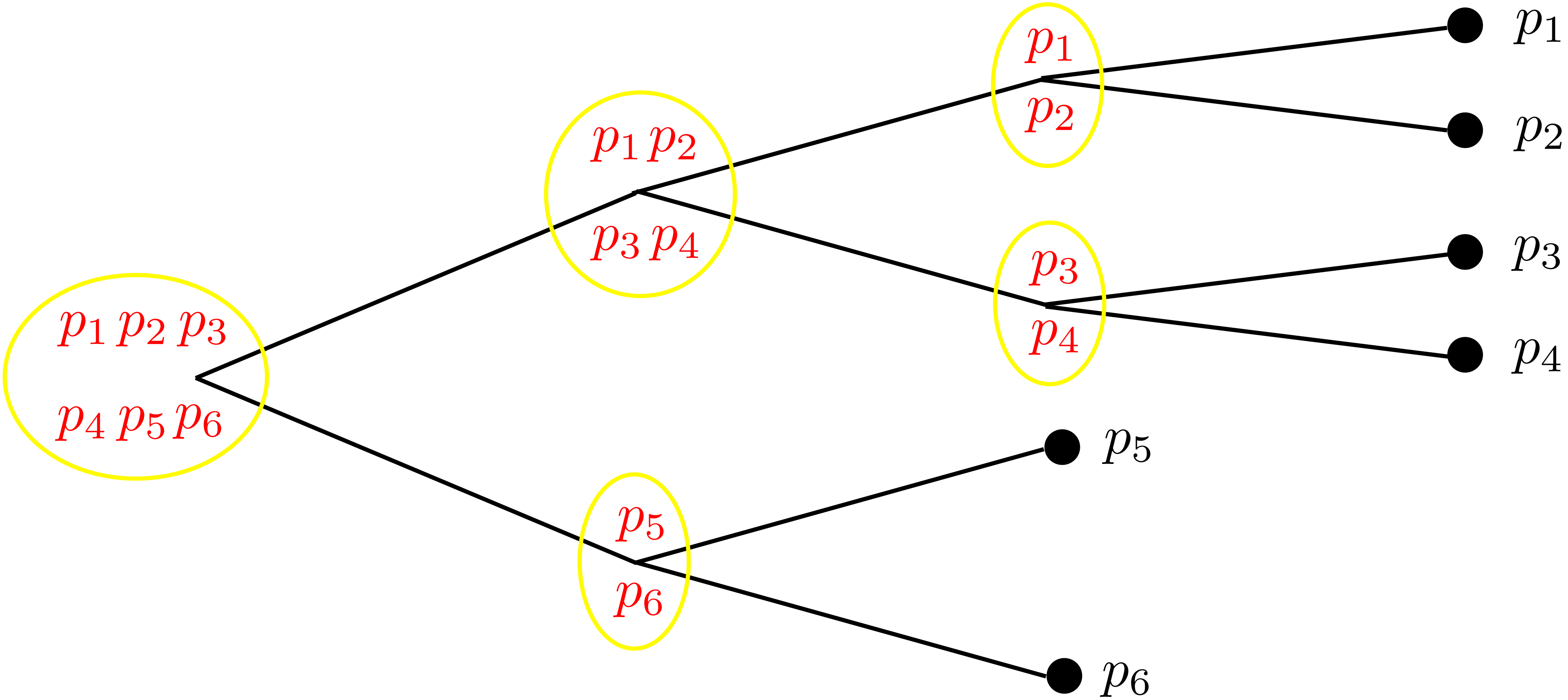
Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

$$\boxed{L = 3p_1 + 3p_2 + 3p_3 + 3p_4 + 2p_5 + 2p_6}$$



Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

$$\boxed{L = 3p_1 + 3p_2 + 3p_3 + 3p_4 + 2p_5 + 2p_6}$$



Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

Proof

Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

Proof

1. Define

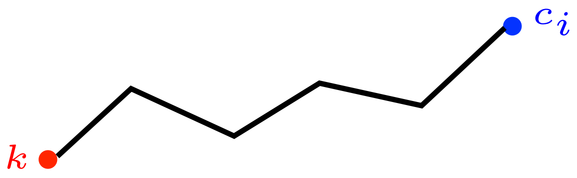
$$a_{ki} = \begin{cases} 1 & \text{if leaf } c_i \text{ is a descendent of internal node } k \\ 0 & \text{otherwise.} \end{cases}$$

Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

Proof

1. Define

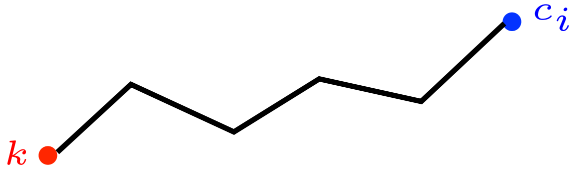
$$a_{ki} = \begin{cases} 1 & \text{if leaf } c_i \text{ is a descendent of internal node } k \\ 0 & \text{otherwise.} \end{cases}$$



Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

Proof

1. Define



$$a_{ki} = \begin{cases} 1 & \text{if leaf } c_i \text{ is a descendent of internal node } k \\ 0 & \text{otherwise.} \end{cases}$$

2. Then

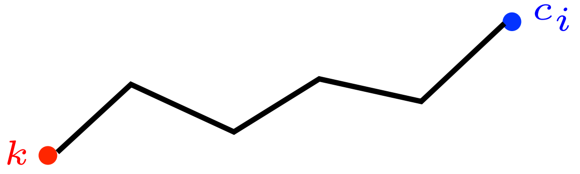
$$l_i = \sum_{k \in \mathcal{I}} a_{ki},$$

Lemma 4.20 $L = \sum_{k\in\mathcal{I}} q_k$.

Proof

1. Define

$$a_{ki} = \begin{cases} 1 & \text{if leaf } c_i \text{ is a descendent of internal node } k \\ 0 & \text{otherwise.} \end{cases}$$



2. Then

$$l_i = \sum_{k\in\mathcal{I}} a_{ki},$$

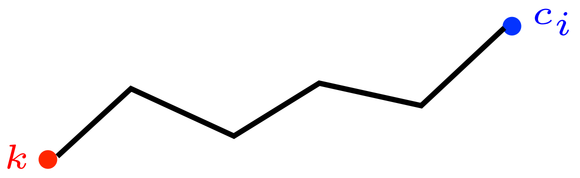
because there are exactly l_i internal nodes of which c_i is a descendent if the order of c_i is l_i.

Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

Proof

1. Define

$$a_{ki} = \begin{cases} 1 & \text{if leaf } c_i \text{ is a descendent of internal node } k \\ 0 & \text{otherwise.} \end{cases}$$



2. Then

$$l_i = \sum_{k \in \mathcal{I}} a_{ki},$$

because there are exactly l_i internal nodes of which c_i is a descendent if the order of c_i is l_i.

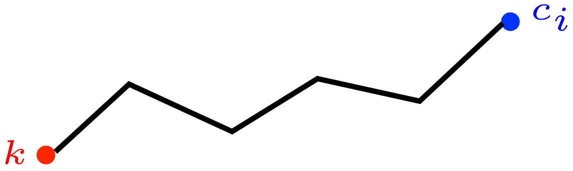
3. On the other hand,

Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

Proof

1. Define

$$a_{ki} = \begin{cases} 1 & \text{if leaf } c_i \text{ is a descendent of internal node } k \\ 0 & \text{otherwise.} \end{cases}$$



2. Then

$$l_i = \sum_{k \in \mathcal{I}} a_{ki},$$

because there are exactly l_i internal nodes of which c_i is a descendent if the order of c_i is l_i.

3. On the other hand,

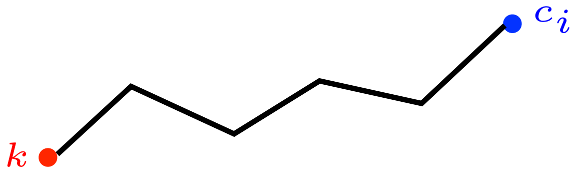
$$q_k = \sum_i a_{ki} p_i.$$

Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

Proof

1. Define

$$a_{ki} = \begin{cases} 1 & \text{if leaf } c_i \text{ is a descendent of internal node } k \\ 0 & \text{otherwise.} \end{cases}$$



2. Then

$$l_i = \sum_{k \in \mathcal{I}} a_{ki},$$

because there are exactly l_i internal nodes of which c_i is a descendent if the order of c_i is l_i.

3. On the other hand,

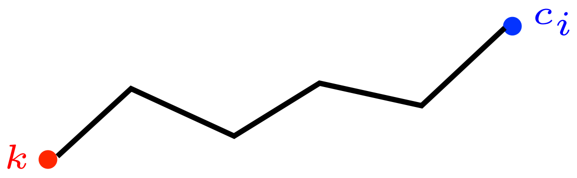
$$q_k = \sum_i a_{ki} p_i.$$

Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

Proof

1. Define

$$a_{ki} = \begin{cases} 1 & \text{if leaf } c_i \text{ is a descendent of internal node } k \\ 0 & \text{otherwise.} \end{cases}$$



2. Then

$$l_i = \sum_{k \in \mathcal{I}} a_{ki},$$

because there are exactly l_i internal nodes of which c_i is a descendent if the order of c_i is l_i.

3. On the other hand,

$$q_k = \sum_i a_{ki} p_i.$$

4. Then

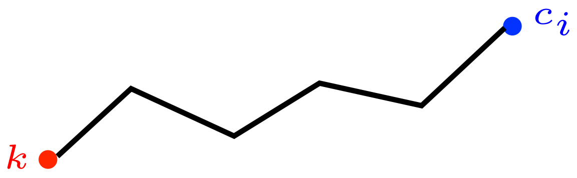
$$L = \sum_i p_i l_i$$

Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

Proof

1. Define

$$a_{ki} = \begin{cases} 1 & \text{if leaf } c_i \text{ is a descendent of internal node } k \\ 0 & \text{otherwise.} \end{cases}$$



2. Then

$$l_i = \sum_{k \in \mathcal{I}} a_{ki},$$

because there are exactly l_i internal nodes of which c_i is a descendent if the order of c_i is l_i.

3. On the other hand,

$$q_k = \sum_i a_{ki} p_i.$$

4. Then

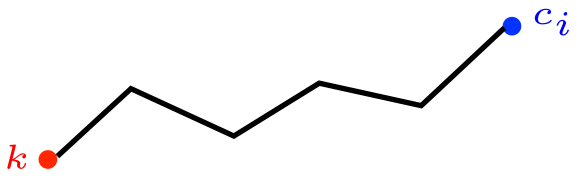
$$L = \sum_i p_i \underline{l_i}$$

Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

Proof

1. Define

$$a_{ki} = \begin{cases} 1 & \text{if leaf } c_i \text{ is a descendent of internal node } k \\ 0 & \text{otherwise.} \end{cases}$$



2. Then

$$l_i = \sum_{k \in \mathcal{I}} a_{ki},$$

because there are exactly l_i internal nodes of which c_i is a descendent if the order of c_i is l_i.

3. On the other hand,

$$q_k = \sum_i a_{ki} p_i .$$

4. Then

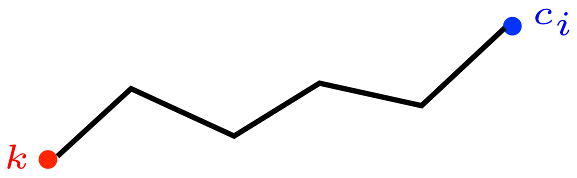
$$\begin{aligned} L &= \sum_i p_i \underline{l_i} \\ &= \sum_i p_i \underline{\sum_{k \in \mathcal{I}} a_{ki}} \end{aligned}$$

Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

Proof

1. Define

$$a_{ki} = \begin{cases} 1 & \text{if leaf } c_i \text{ is a descendent of internal node } k \\ 0 & \text{otherwise.} \end{cases}$$



2. Then

$$l_i = \sum_{k \in \mathcal{I}} a_{ki},$$

because there are exactly l_i internal nodes of which c_i is a descendent if the order of c_i is l_i.

3. On the other hand,

$$q_k = \sum_i a_{ki} p_i.$$

4. Then

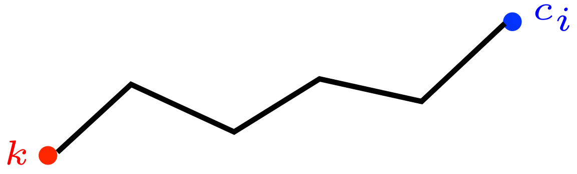
$$\begin{aligned} L &= \sum_i p_i l_i \\ &= \sum_i p_i \sum_{k \in \mathcal{I}} a_{ki} \end{aligned}$$

Lemma 4.20 $L = \sum_{k\in\mathcal{I}} q_k$.

Proof

1. Define

$$a_{ki} = \begin{cases} 1 & \text{if leaf } c_i \text{ is a descendent of internal node } k \\ 0 & \text{otherwise.} \end{cases}$$



2. Then

$$l_i = \sum_{k\in\mathcal{I}} a_{ki},$$

because there are exactly l_i internal nodes of which c_i is a descendent if the order of c_i is l_i.

3. On the other hand,

$$q_k = \sum_i a_{ki} p_i.$$

4. Then

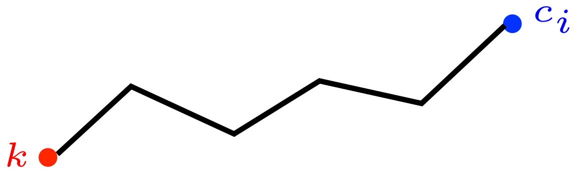
$$\begin{aligned} L &= \sum_i p_i l_i \\ &= \sum_i p_i \sum_{k\in\mathcal{I}} a_{ki} \\ &= \sum_{k\in\mathcal{I}} \sum_i p_i a_{ki} \end{aligned}$$

Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

Proof

1. Define

$$a_{ki} = \begin{cases} 1 & \text{if leaf } c_i \text{ is a descendent of internal node } k \\ 0 & \text{otherwise.} \end{cases}$$



2. Then

$$l_i = \sum_{k \in \mathcal{I}} a_{ki},$$

because there are exactly l_i internal nodes of which c_i is a descendent if the order of c_i is l_i.

3. On the other hand,

$$q_k = \sum_i a_{ki} p_i.$$

4. Then

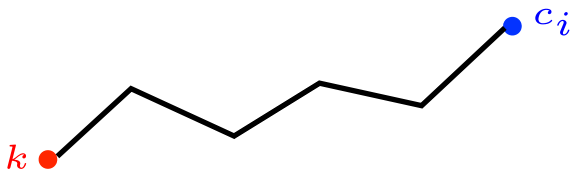
$$\begin{aligned} L &= \sum_i p_i l_i \\ &= \sum_i p_i \sum_{k \in \mathcal{I}} a_{ki} \\ &= \sum_{k \in \mathcal{I}} \underline{\sum_i p_i a_{ki}} \end{aligned}$$

Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

Proof

1. Define

$$a_{ki} = \begin{cases} 1 & \text{if leaf } c_i \text{ is a descendent of internal node } k \\ 0 & \text{otherwise.} \end{cases}$$



2. Then

$$l_i = \sum_{k \in \mathcal{I}} a_{ki},$$

because there are exactly l_i internal nodes of which c_i is a descendent if the order of c_i is l_i.

3. On the other hand,

$$q_k = \sum_i a_{ki} p_i.$$

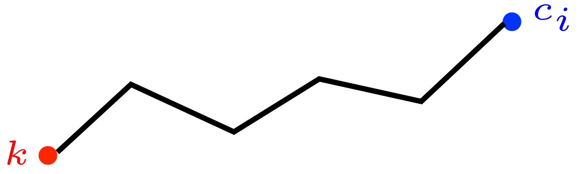
4. Then

$$\begin{aligned} L &= \sum_i p_i l_i \\ &= \sum_i p_i \sum_{k \in \mathcal{I}} a_{ki} \\ &= \sum_{k \in \mathcal{I}} \underline{\sum_i p_i a_{ki}} \\ &= \sum_{k \in \mathcal{I}} \underline{q_k}, \end{aligned}$$

Lemma 4.20 $L = \sum_{k \in \mathcal{I}} q_k$.

Proof

1. Define



$$a_{ki} = \begin{cases} 1 & \text{if leaf } c_i \text{ is a descendent of internal node } k \\ 0 & \text{otherwise.} \end{cases}$$

2. Then

$$l_i = \sum_{k \in \mathcal{I}} a_{ki},$$

because there are exactly l_i internal nodes of which c_i is a descendent if the order of c_i is l_i.

3. On the other hand,

$$q_k = \sum_i a_{ki} p_i.$$

4. Then

$$\begin{aligned} L &= \sum_i p_i l_i \\ &= \sum_i p_i \sum_{k \in \mathcal{I}} a_{ki} \\ &= \sum_{k \in \mathcal{I}} \sum_i p_i a_{ki} \\ &= \sum_{k \in \mathcal{I}} q_k, \end{aligned}$$

proving the lemma.

Local Redundancy

Local Redundancy

- Define the local redundancy of an internal node k by

$$r_k = q_k(1 - h_k).$$

Local Redundancy

- Define the local redundancy of an internal node k by

$$r_k = \underline{q_k}(1 - h_k).$$

q_k: reaching probability of internal node k

Local Redundancy

- Define the local redundancy of an internal node k by

$$r_k = q_k(1 - \underline{h_k}).$$

q_k: reaching probability of internal node k

h_k: conditional entropy of internal node k

Local Redundancy

- Define the local redundancy of an internal node k by

$$r_k = q_k(1 - h_k).$$

Local Redundancy

- Define the local redundancy of an internal node k by

$$r_k = q_k(1 - h_k).$$

- This quantity is local to node k in the sense that it depends only on the branching probabilities of node k.

Local Redundancy

- Define the local redundancy of an internal node k by
$$r_k = q_k(1 - h_k).$$
- This quantity is local to node k in the sense that it depends only on the branching probabilities of node k.
- $r_k = 0$ if and only if
$$\tilde{p}_{k,j} = \frac{q_k}{D} \quad \text{for all } j,$$
i.e., if and only if the internal node k is balanced.

Local Redundancy

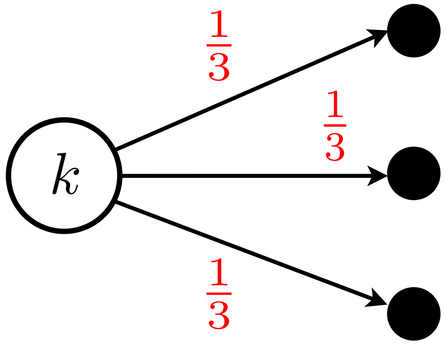
- Define the local redundancy of an internal node k by

$$r_k = q_k(1 - h_k).$$

- This quantity is local to node k in the sense that it depends only on the branching probabilities of node k.
- $r_k = 0$ if and only if

$$\tilde{p}_{k,j} = \frac{q_k}{D} \quad \text{for all } j,$$

i.e., if and only if the internal node k is balanced.



Local Redundancy

- Define the local redundancy of an internal node k by

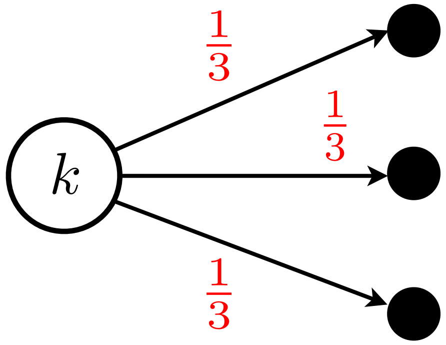
$$r_k = q_k(1 - h_k).$$

- This quantity is local to node k in the sense that it depends only on the branching probabilities of node k.

- $r_k = 0$ if and only if

$$\tilde{p}_{k,j} = \frac{q_k}{D} \quad \text{for all } j,$$

i.e., if and only if the internal node k is balanced.



Local Redundancy

- Define the local redundancy of an internal node k by

$$r_k = q_k(1 - h_k).$$

- This quantity is local to node k in the sense that it depends only on the branching probabilities of node k.

- $r_k = 0$ if and only if

$$\tilde{p}_{k,j} = \frac{q_k}{D} \quad \text{for all } j,$$

  i.e., if and only if the internal node k is balanced.

- $r_k \geq 0$ because $h_k \leq 1$.

Theorem 4.21 (Local Redundancy Theorem) Let R be the redundancy of a D-ary prefix code for a source random variable X. Then

$$R = \sum_{k \in \mathcal{I}} r_k.$$

Proof

Theorem 4.21 (Local Redundancy Theorem) Let R be the redundancy of a D-ary prefix code for a source random variable X. Then

$$R = \sum_{k \in \mathcal{I}} r_k.$$

Proof

$$R = L - H_D(X)$$

Theorem 4.21 (Local Redundancy Theorem) Let R be the redundancy of a D-ary prefix code for a source random variable X. Then

$$R = \sum_{k \in \mathcal{I}} r_k .$$

Proof

$$R \quad = \quad L - H_D(X)$$

Theorem 4.21 (Local Redundancy Theorem) Let R be the redundancy of a D-ary prefix code for a source random variable X. Then

$$R = \sum_{k \in \mathcal{I}} r_k .$$

Proof

$$\begin{aligned} R &= \underline{L} - H_D(X) \\ &= \underline{\sum_{k \in \mathcal{I}} q_k} - \sum_{k \in \mathcal{I}} q_k h_k \end{aligned}$$

Theorem 4.21 (Local Redundancy Theorem) Let R be the redundancy of a D-ary prefix code for a source random variable X. Then

$$R = \sum_{k \in \mathcal{I}} r_k .$$

Proof

$$\begin{aligned} R &= L - \underline{H_D(X)} \\ &= \sum_{k \in \mathcal{I}} q_k - \sum_{k \in \mathcal{I}} q_k h_k \end{aligned}$$

Theorem 4.21 (Local Redundancy Theorem) Let R be the redundancy of a D-ary prefix code for a source random variable X. Then

$$R = \sum_{k \in \mathcal{I}} r_k.$$

Proof

$$\begin{aligned} R &= L - \underline{H_D(X)} \\ &= \sum_{k \in \mathcal{I}} q_k - \underline{\sum_{k \in \mathcal{I}} q_k h_k} \end{aligned}$$

Theorem 4.21 (Local Redundancy Theorem) Let R be the redundancy of a D-ary prefix code for a source random variable X. Then

$$R = \sum_{k \in \mathcal{I}} r_k.$$

Proof

$$\begin{aligned} R &= L - H_D(X) \\ &= \sum_{k \in \mathcal{I}} \underline{q_k} - \sum_{k \in \mathcal{I}} \underline{q_k h_k} \end{aligned}$$

Theorem 4.21 (Local Redundancy Theorem) Let R be the redundancy of a D-ary prefix code for a source random variable X. Then

$$R = \sum_{k \in \mathcal{I}} r_k.$$

Proof

$$\begin{aligned} R &= L - H_D(X) \\ &= \sum_{k \in \mathcal{I}} \underline{q_k} - \sum_{k \in \mathcal{I}} \underline{q_k h_k} \\ &= \sum_{k \in \mathcal{I}} \underline{q_k (1 - h_k)} \end{aligned}$$

Theorem 4.21 (Local Redundancy Theorem) Let R be the redundancy of a D-ary prefix code for a source random variable X. Then

$$R = \sum_{k \in \mathcal{I}} r_k.$$

Proof

$$\begin{aligned} R &= L - H_D(X) \\ &= \sum_{k \in \mathcal{I}} q_k - \sum_{k \in \mathcal{I}} q_k h_k \\ &= \sum_{k \in \mathcal{I}} \underline{q_k(1 - h_k)} \\ &= \sum_{k \in \mathcal{I}} \underline{r_k}. \end{aligned}$$

Theorem 4.21 (Local Redundancy Theorem) Let R be the redundancy of a D-ary prefix code for a source random variable X. Then

$$R = \sum_{k \in \mathcal{I}} r_k.$$

Proof

$$\begin{aligned} R &= L - H_D(X) \\ &= \sum_{k \in \mathcal{I}} q_k - \sum_{k \in \mathcal{I}} q_k h_k \\ &= \sum_{k \in \mathcal{I}} q_k (1 - h_k) \\ &= \sum_{k \in \mathcal{I}} r_k. \end{aligned}$$

Corollary 4.22 (Entropy Bound for Prefix Code) Let R be the redundancy of a prefix code. Then $R \geq 0$ with equality if and only if all the internal nodes in the code tree are balanced.

Proof

Corollary 4.22 (Entropy Bound for Prefix Code) Let R be the redundancy of a prefix code. Then $R \geq 0$ with equality if and only if all the internal nodes in the code tree are balanced.

Proof

1. Consider

$$R = \sum_{k \in \mathcal{I}} r_k .$$

Corollary 4.22 (Entropy Bound for Prefix Code) Let R be the redundancy of a prefix code. Then $R \geq 0$ with equality if and only if all the internal nodes in the code tree are balanced.

Proof

1. Consider
$$R = \sum_{k \in \mathcal{I}} r_k.$$

2. $R \geq 0$ because $r_k \geq 0$ for all internal nodes k.

Corollary 4.22 (Entropy Bound for Prefix Code) Let R be the redundancy of a prefix code. Then $R \geq 0$ with equality if and only if all the internal nodes in the code tree are balanced.

Proof

1. Consider
$$R = \sum_{k \in \mathcal{I}} r_k.$$
2. $R \geq 0$ because $r_k \geq 0$ for all internal nodes k.
3. $R = 0$ if and only if $r_k = 0$ for all k, which means that all the internal nodes in the code tree are balanced.

Remarks

Remarks

- The entropy bound says that for a D-ary uniquely decodable code $\mathcal{C}$,

$$H_D(X) \leq L.$$

Remarks

- The entropy bound says that for a D-ary uniquely decodable code $\mathcal{C}$,

$$H_D(X) \leq L.$$

- This makes sense because intuitively each D-ary symbol can carry at most 1 D-it of information.

Remarks

- The entropy bound says that for a D-ary uniquely decodable code $\mathcal{C}$,

$$H_D(X) \leq L.$$

- This makes sense because intuitively each D-ary symbol can carry at most 1 D-it of information.
- Therefore, when the entropy bound is tight, each code symbol has to carry exactly one D-it of information.

Remarks

- The entropy bound says that for a D-ary uniquely decodable code $\mathcal{C}$,

$$H_D(X) \leq L.$$

- This makes sense because intuitively each D-ary symbol can carry at most 1 D-it of information.
- Therefore, when the entropy bound is tight, each code symbol has to carry exactly one D-it of information.
- In the context of a prefix code, the interpretation is as follows:

Remarks

- The entropy bound says that for a D-ary uniquely decodable code $\mathcal{C}$,

$$H_D(X) \leq L.$$

- This makes sense because intuitively each D-ary symbol can carry at most 1 D-it of information.
- Therefore, when the entropy bound is tight, each code symbol has to carry exactly one D-it of information.
- In the context of a prefix code, the interpretation is as follows:

    1. Consider revealing a random codeword one symbol after another.

Remarks

- The entropy bound says that for a D-ary uniquely decodable code $\mathcal{C}$,

$$H_D(X) \leq L.$$

- This makes sense because intuitively each D-ary symbol can carry at most 1 D-it of information.

- Therefore, when the entropy bound is tight, each code symbol has to carry exactly one D-it of information.

- In the context of a prefix code, the interpretation is as follows:

  1. Consider revealing a random codeword one symbol after another.
  2. Corollary 4.22 states that in order for the entropy bound to be tight, all the internal nodes in the code tree must be balanced.

Remarks

- The entropy bound says that for a D-ary uniquely decodable code $\mathcal{C}$,

$$H_D(X) \leq L.$$

- This makes sense because intuitively each D-ary symbol can carry at most 1 D-it of information.
- Therefore, when the entropy bound is tight, each code symbol has to carry exactly one D-it of information.
- In the context of a prefix code, the interpretation is as follows:

  1. Consider revealing a random codeword one symbol after another.
  2. Corollary 4.22 states that in order for the entropy bound to be tight, all the internal nodes in the code tree must be balanced.
  3. That is, as long as the codeword is not completed, the next code symbol to be revealed always carries one D-it of information because it is distributed uniformly on the alphabet.

A Lower Bound on *R*

A Lower Bound on *R*

- Consider

$$R = \sum_{k \in \mathcal{I}} r_k$$

A Lower Bound on *R*

- Consider

$$R = \sum_{k \in \underline{\mathcal{I}}} r_k$$

A Lower Bound on *R*

- Consider

$$R = \sum_{k \in \mathcal{I}} r_k \geq \sum_{k \in \mathcal{I}'} r_k$$

for any subset $\mathcal{I}'$ of $\mathcal{I}$.

A Lower Bound on *R*

- Consider
$$R = \sum_{k\in\mathcal{I}} r_k \geq \sum_{k\in\mathcal{I}'} r_k$$
for any subset $\mathcal{I}'$ of $\mathcal{I}$.
- If we can compute r_k for all $k \in \mathcal{I}'$, we can compute the lower bound
$$R \geq \sum_{k\in\mathcal{I}'} r_k.$$

A Lower Bound on *R*

- Consider

$$R = \sum_{k \in \mathcal{I}} r_k \geq \sum_{k \in \mathcal{I}'} r_k$$

  for any subset $\mathcal{I}'$ of $\mathcal{I}$.

- If we can compute r_k for all $k \in \mathcal{I}'$, we can compute the lower bound

$$R \geq \sum_{k \in \mathcal{I}'} r_k.$$

Example 4.23

Example 4.23

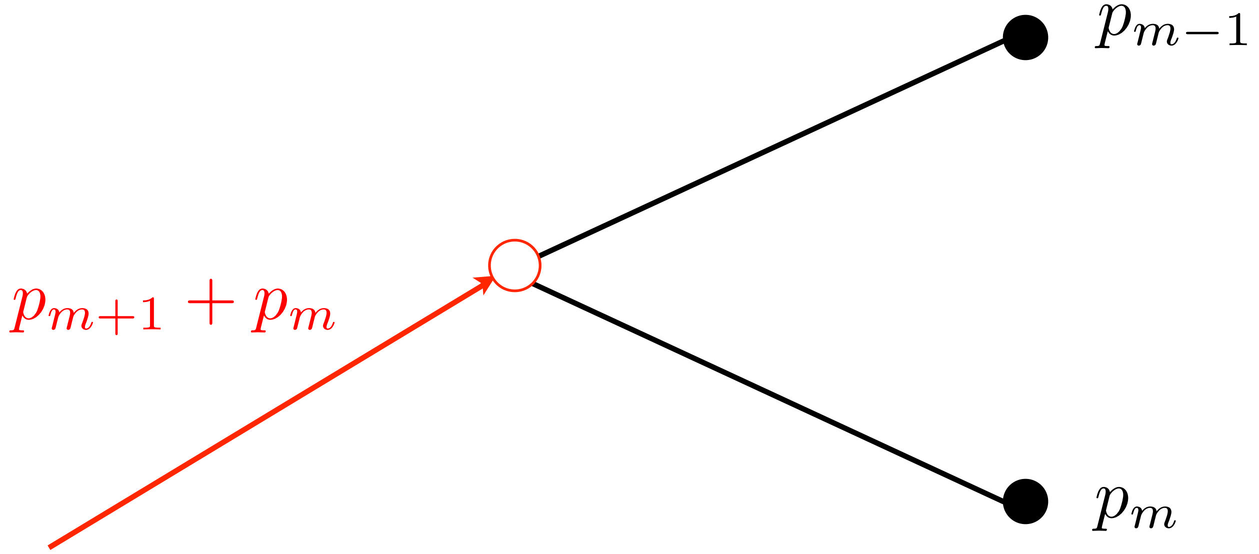
- In the binary Huffman procedure, the two smallest probabilities p_{m-1} and p_m are merged to form an internal node of the code tree.

Example 4.23

- In the binary Huffman procedure, the two smallest probabilities p_{m-1} and p_m are merged to form an internal node of the code tree.
- The reaching probability of this internal node is $p_{m-1} + p_m$.

Example 4.23

- In the binary Huffman procedure, the two smallest probabilities p_{m-1} and p_m are merged to form an internal node of the code tree.
- The reaching probability of this internal node is $p_{m-1} + p_m$.



Example 4.23

- In the binary Huffman procedure, the two smallest probabilities p_{m-1} and p_m are merged to form an internal node of the code tree.

- The reaching probability of this internal node is $p_{m-1} + p_m$.

Example 4.23

- In the binary Huffman procedure, the two smallest probabilities p_{m-1} and p_m are merged to form an internal node of the code tree.
- The reaching probability of this internal node is $p_{m-1} + p_m$.
- The conditional entropy of this internal node is

$$H_2\left(\left\{\frac{p_{m-1}}{p_{m-1}+p_m}, \frac{p_m}{p_{m-1}+p_m}\right\}\right).$$

Example 4.23

- In the binary Huffman procedure, the two smallest probabilities p_{m-1} and p_m are merged to form an internal node of the code tree.

- The reaching probability of this internal node is $p_{m-1} + p_m$.

- The conditional entropy of this internal node is

$$H_2\left(\left\{\frac{p_{m-1}}{p_{m-1}+p_m}, \frac{p_m}{p_{m-1}+p_m}\right\}\right).$$

- The local redundancy of this internal node is

$$(p_{m-1}+p_m)\left[1 - H_2\left(\left\{\frac{p_{m-1}}{p_{m-1}+p_m}, \frac{p_m}{p_{m-1}+p_m}\right\}\right)\right],$$

  which is a lower bound on R.