NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Tutorial 6 Trie

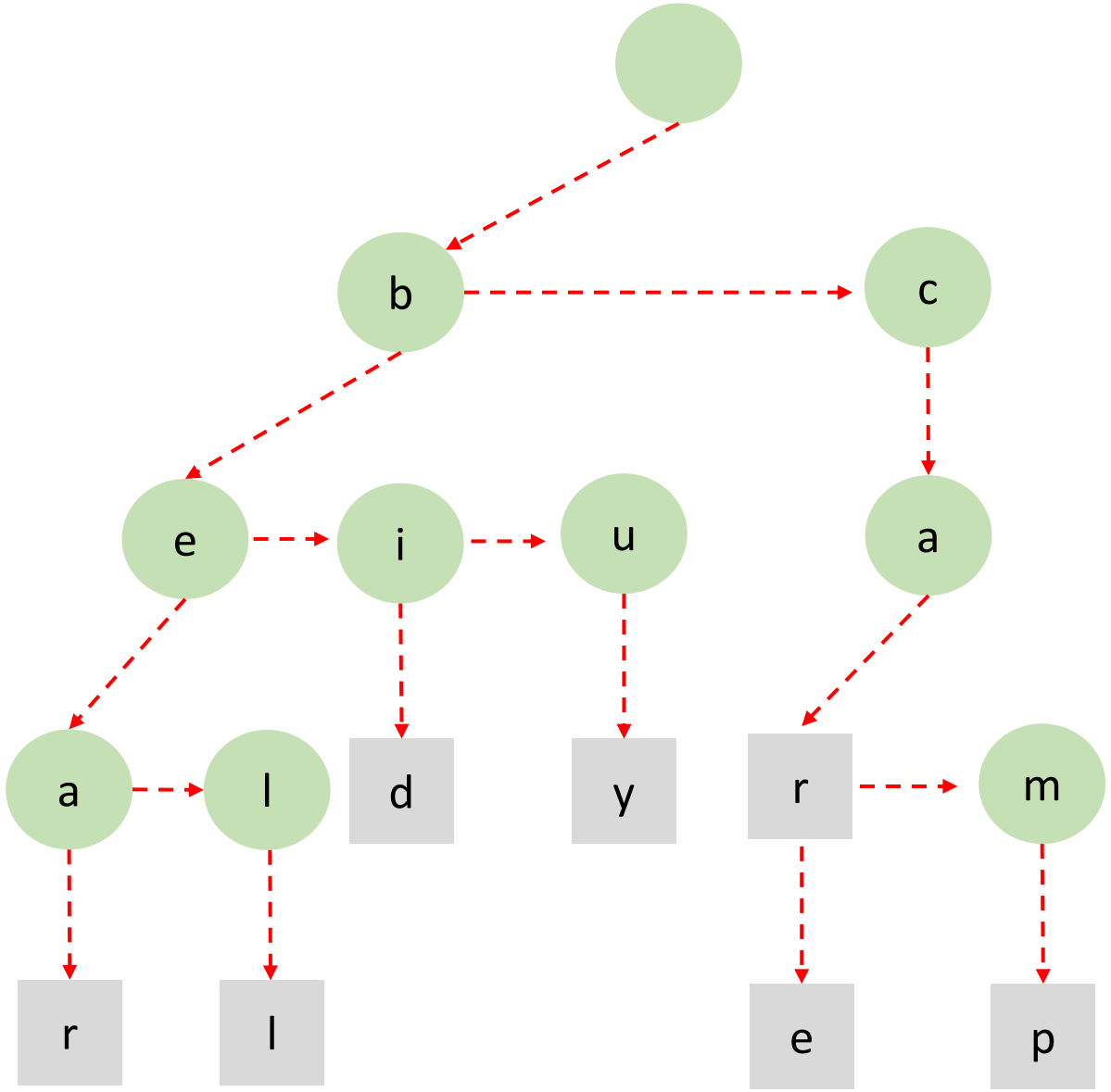
Dr Liu Siyuan
syliu@ntu.edu.sg
N4-02C-72a

Question 1

- You are given a Trie that stores multiple words. Write a function count_words() to count how many words are stored in the Trie. The function prototype is given as follows:

```
def count_words(self, node):
```

Working Example: Print All Words

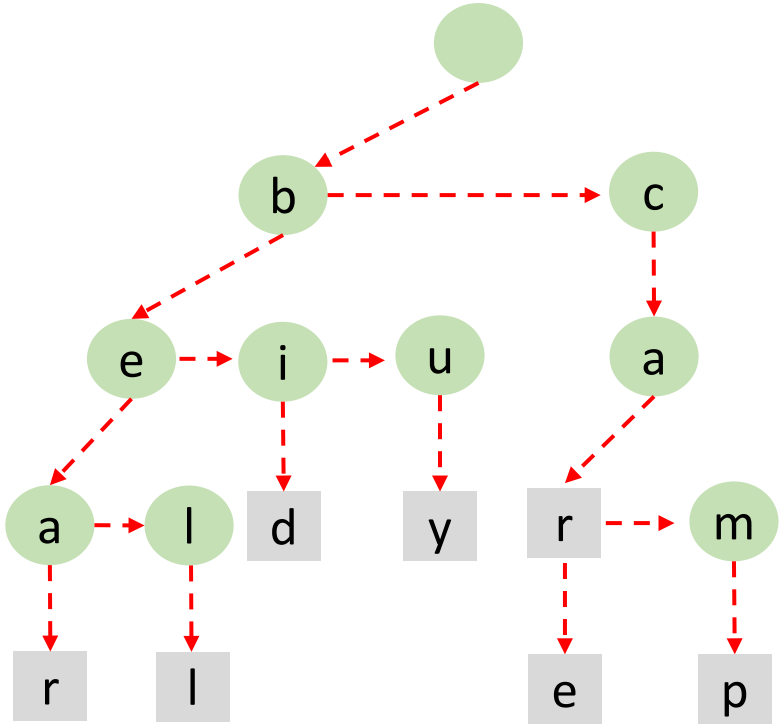



```
def print_all_words_dfs(self, node,
prefix):
    if node.is_end_of_word:
        print(prefix)
    child = node.first_child
    while child:
        self. print_all_words(child,
                              prefix+child.char)
        child = child.next_sibling
```

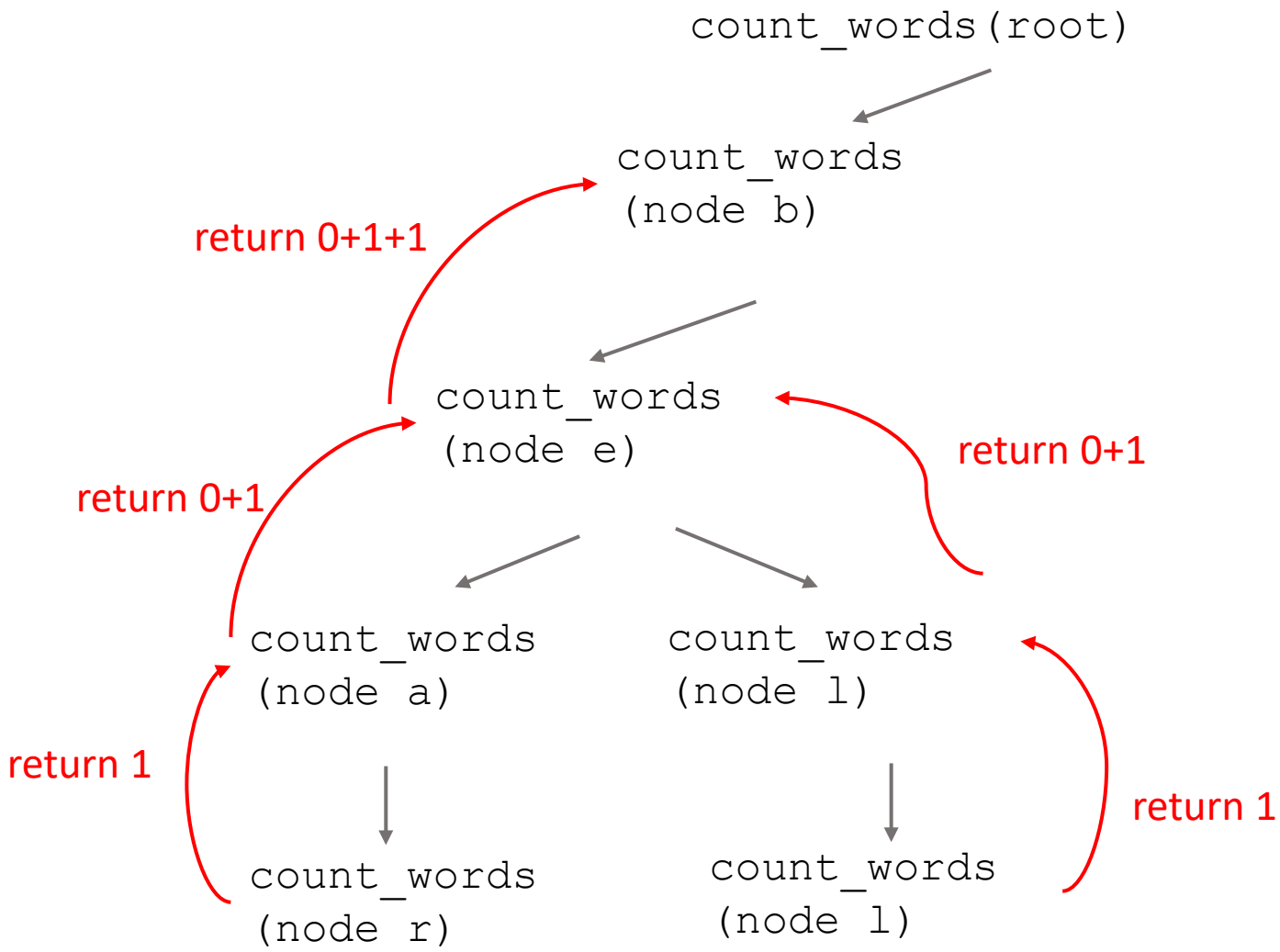
Question 1

```
def print_all_words_dfs(self, node,
prefix):
    if node.is_end_of_word:
        print(prefix)
    child = node.first_child
    while child:
        self.print_all_words(child,
                             prefix+child.char)
        child = child.next_sibling
```

```
def count_words(self, node):
    if node.is_end_of_word:
        count = 1
    else: count = 0
    child = node.first_child
    while child:
        count =
            count+self.count_words(child)
        child = child.next_sibling
    return count
```




```
def count_words(self, node):
    if node.is_end_of_word:
        count = 1
    else: count = 0
    child = node.first_child
    while child:
        count = count+self.count_words(child)
        child = child.next_sibling
    return count
```

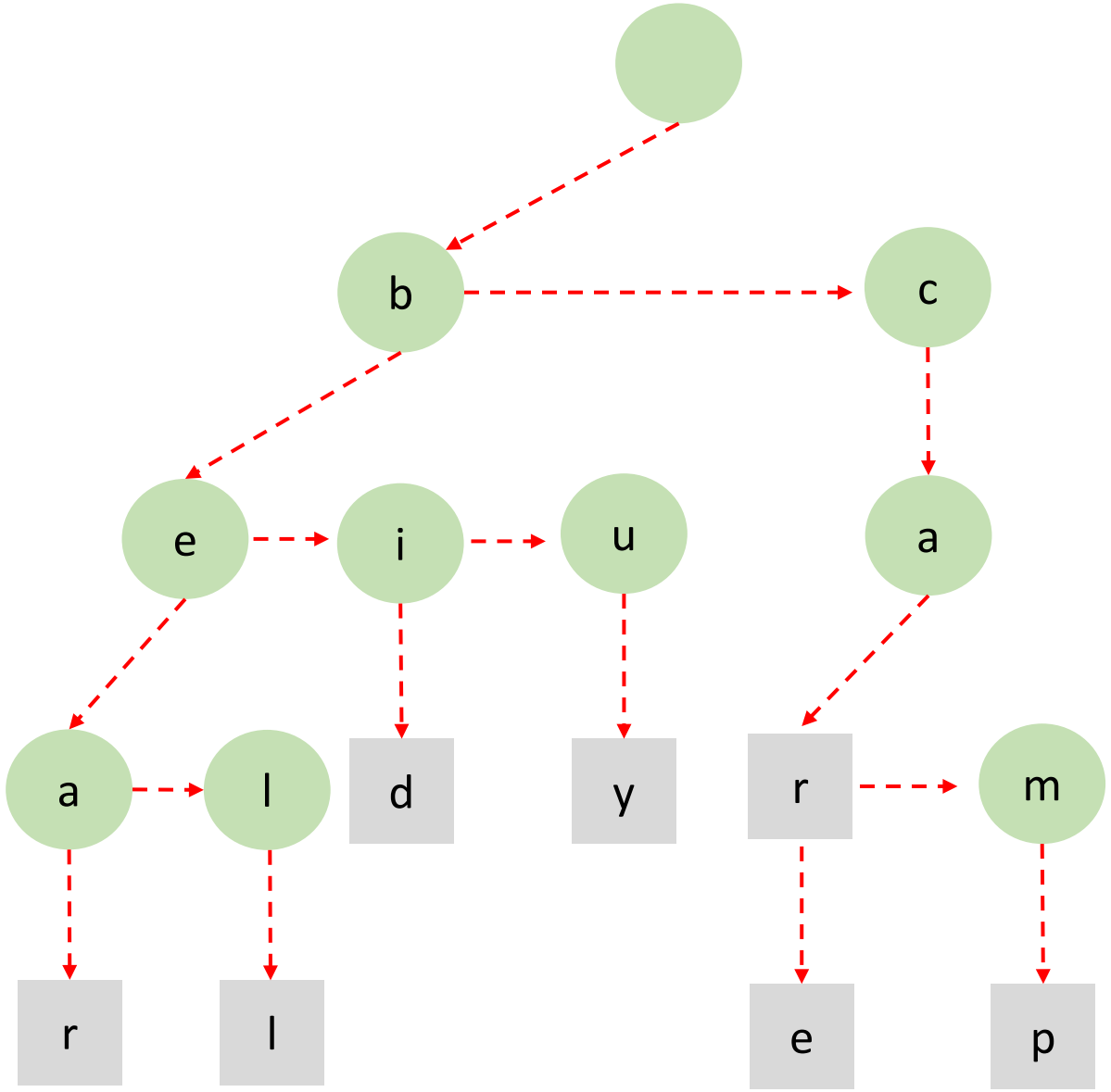


Question 2

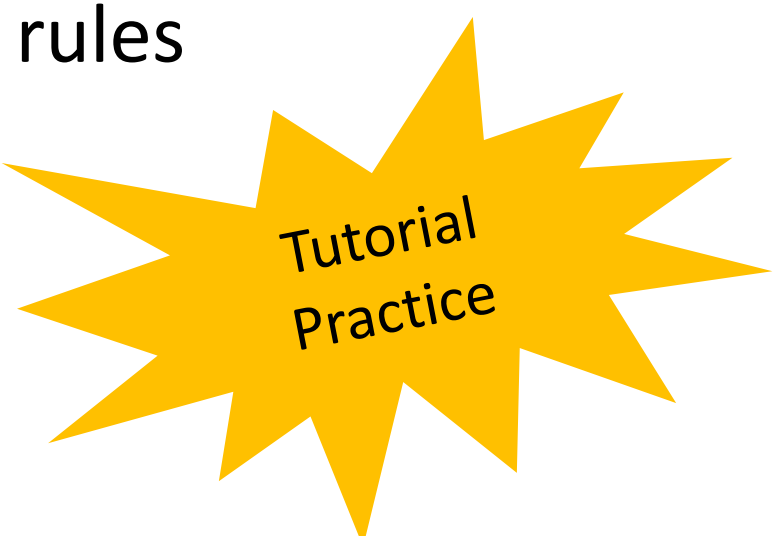
- Given a Trie that stores multiple words, implement a function find_words_with_prefix() that returns all words that start with a given prefix. The function prototype is given as follows:

```
def find_words_with_prefix(self,node,prefix):
```

Application Example: Autocomplete



- Traverse the Trie to the node matching the prefix, e.g., “ca”
- Perform dfs/bfs to collect all complete words
- Return the words based on some rules



Question 2

```
def print_all_words_dfs(self, node, prefix):
    if node.is_end_of_word:
        print(prefix)
    child = node.first_child
    while child:
        self. print_all_words(child,
                        prefix+child.char)
        child = child.next_sibling

def _find_child(self, node, char):
    current = node.child
    while current:
        if current.char == char:
            return current
        current = current.next
    return None
```

```
def find_words_with_prefix(self,node,prefix):
#Use a list to collect the words
    results = []
    node = self.root
#Step 1: Traverse the Trie to the node
matching the prefix
    for char in prefix:
        node = self._find_child(node, char)
        if not node:
            return []
#Step2: Perform dfs to collect all complete
words
    self.collect_all_words_dfs(node, prefix,
                                    results)
    return results
```

Question 2

```
def collect_all_words_dfs(self, node, prefix,
results):
    if node.is_end_of_word:
        results.append(prefix)
    child = node.first_child
    while child:
        self. collect_all_words_dfs(child,
                    prefix+child.char, results)
        child = child.next_sibling
def _find_child(self, node, char):
    current = node.child
    while current:
        if current.char == char:
            return current
        current = current.next
    return None
```

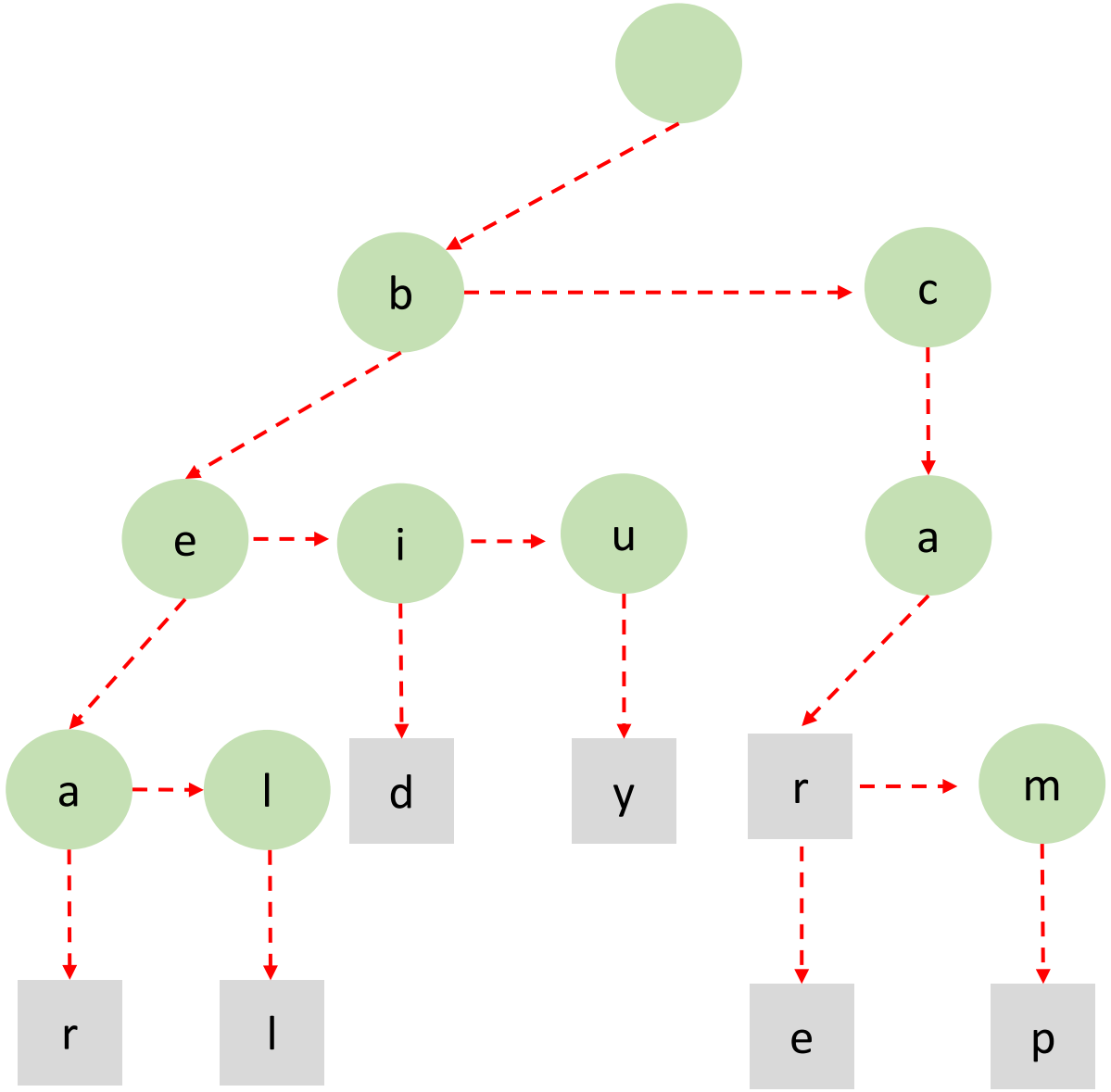
```
def find_words_with_prefix(self,node,prefix):
#Use a list to collect the words
    results = []
    node = self.root
#Step 1: Traverse the Trie to the node
matching the prefix
    for char in prefix:
        node = self._find_child(node, char)
        if not node:
            return []
#Step2: Perform dfs to collect all complete
words
    self.collect_all_words_dfs(node, prefix,
                                results)
    return results
```

Question 3

- Given a Trie storing multiple words, write a function find_shortest_word_with_prefix() that returns the shortest words that starts with a given prefix. If no word starts with the prefix, return None.

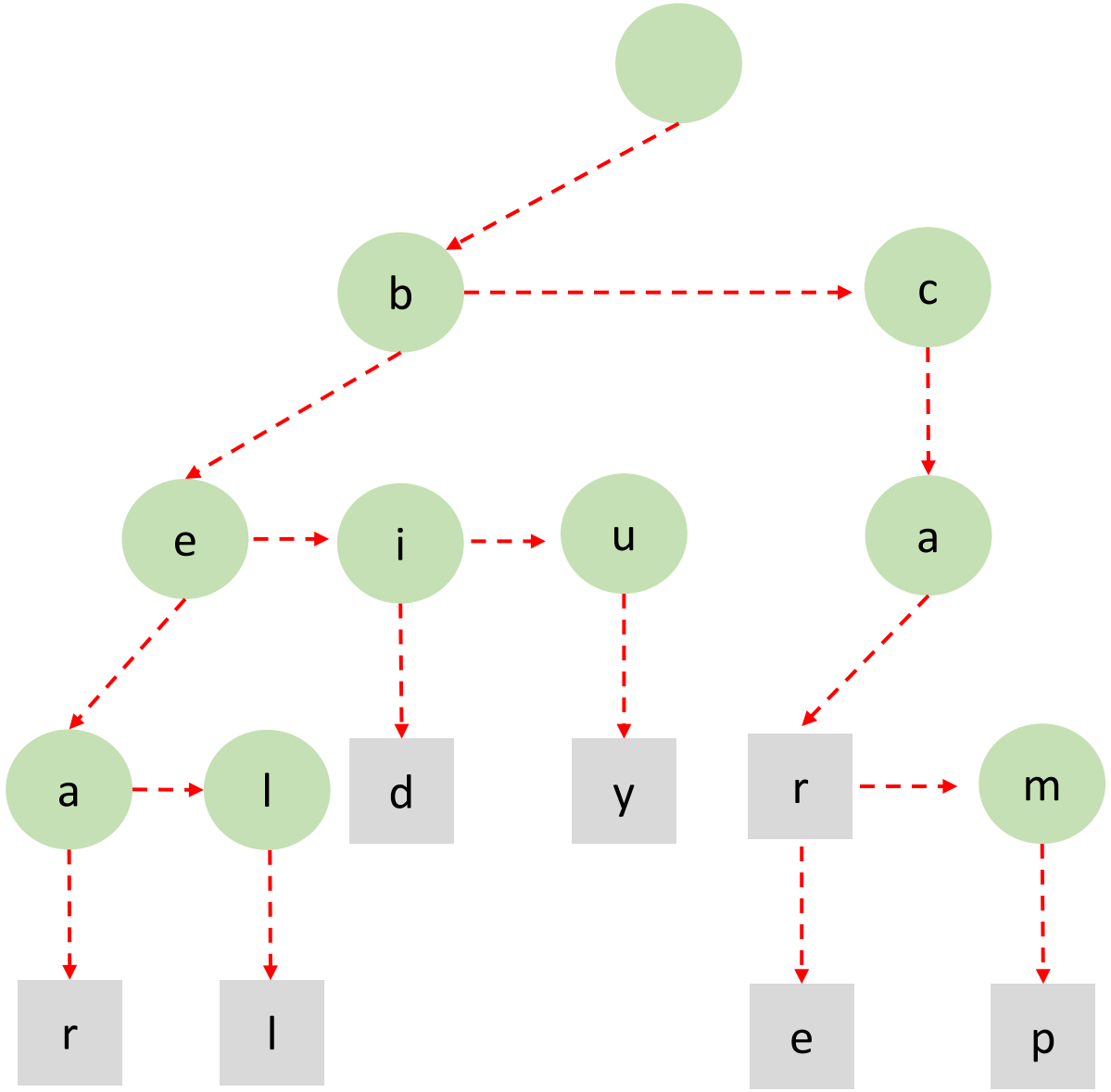
```
def find_shortest_word_with_prefix(self,node,prefix):
```

Question 3



- Traverse the Trie to the node matching the prefix, e.g., “ca”
- Perform bfs from the ending node of the prefix
- The first complete word will be the shortest one.

Working Example: Print All Words (BFS)

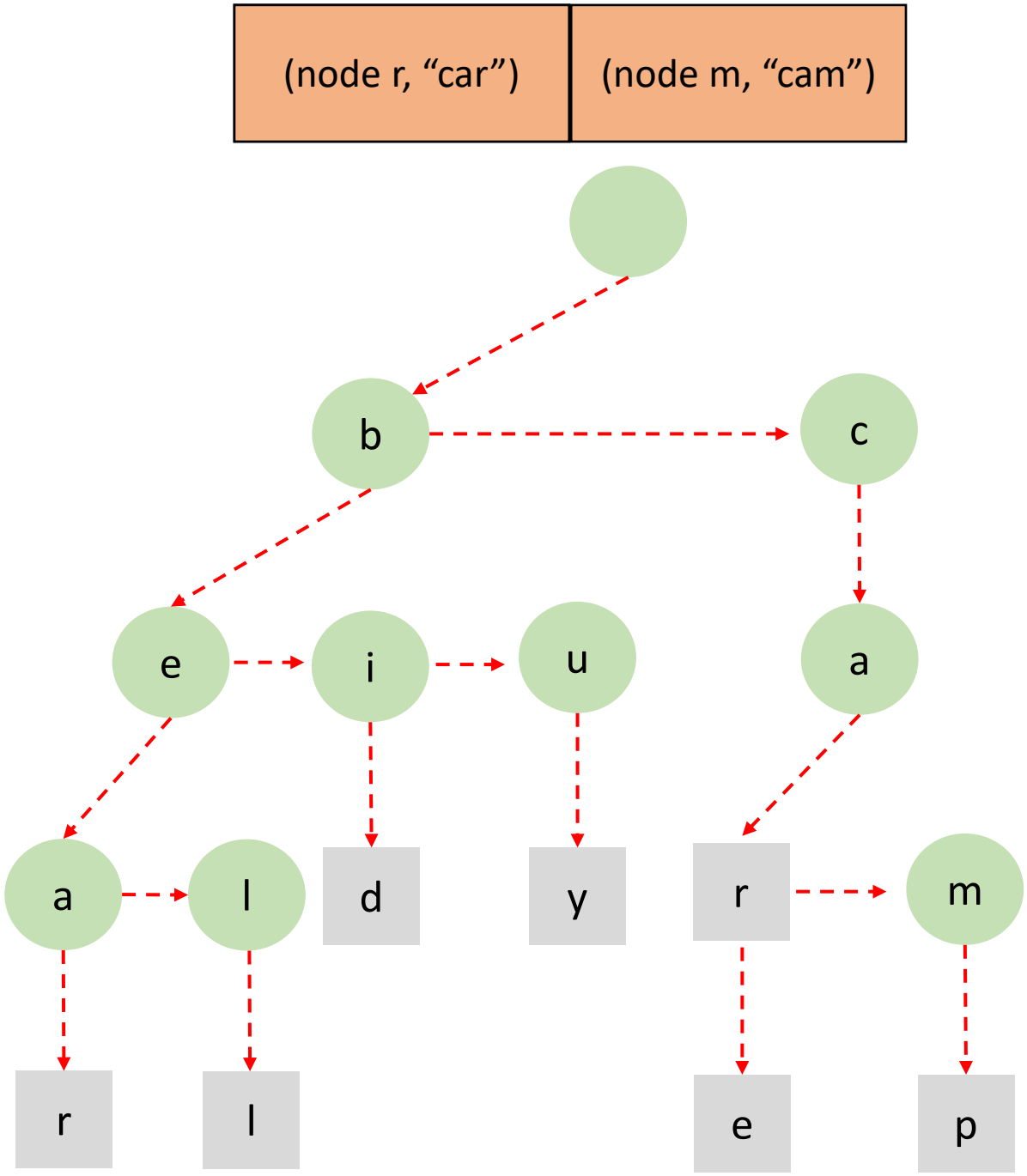



```
def print_all_words_bfs(self):
    queue = Queue()
    queue.enqueue((self.root, ""))
    while not queue.is_empty():
        node, prefix = queue.dequeue()
        if node.is_end_of_word:
            print(prefix)
        child = node.first_child
        while child:
            queue.enqueue((child,
                           prefix + child.char))
            child = child.next_sibling
```

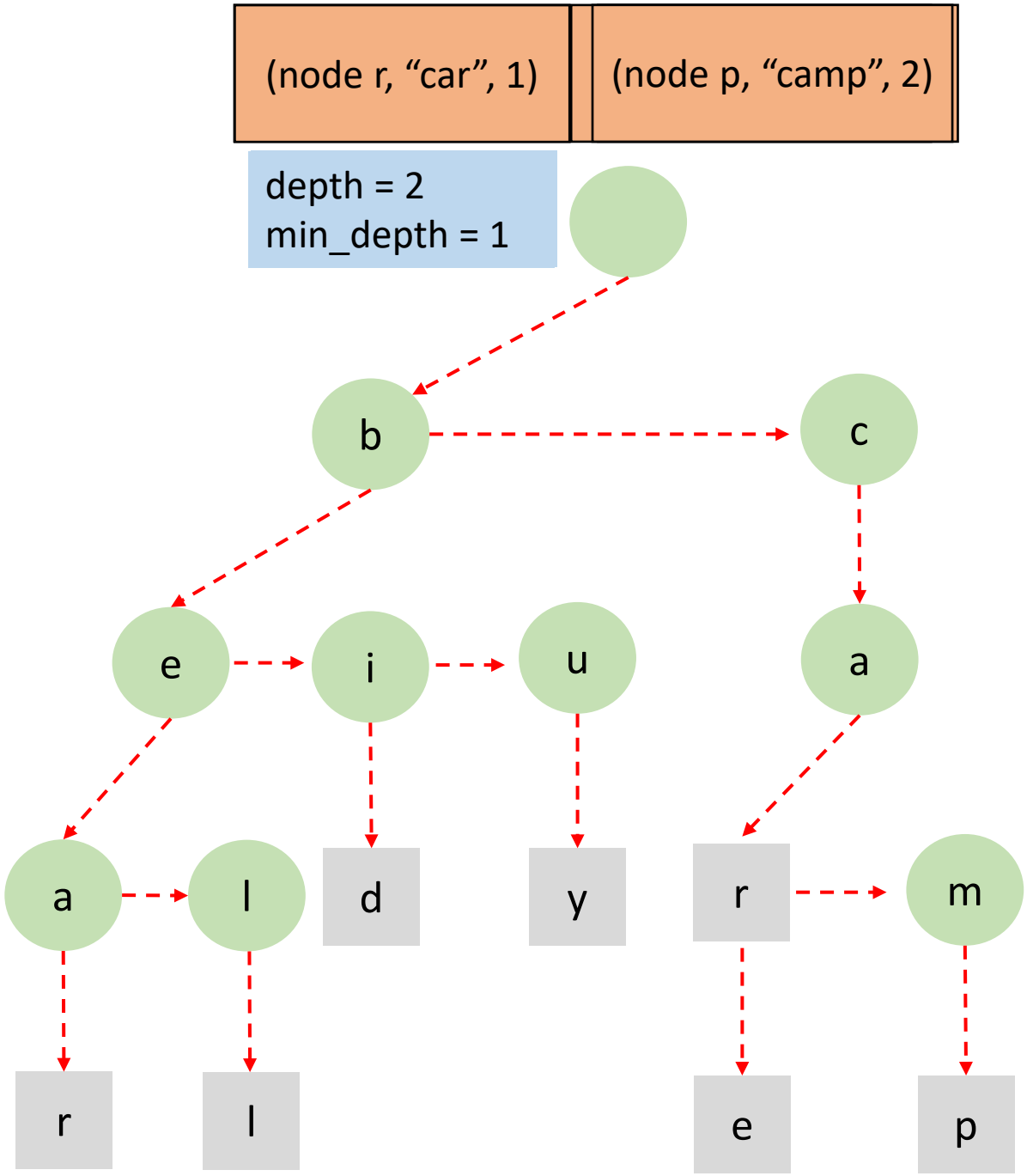
Question 3

```
def print_all_words_bfs(self):
    queue = Queue()
    queue.enqueue((self.root, ""))
    while not queue.is_empty():
        node, prefix = queue.dequeue()
        if node.is_end_of_word:
            print(prefix)
            child = node.first_child
            while child:
                queue.enqueue((child,
                        prefix + child.char))
                child = child.next_sibling
```

```
def find_shortest_word_with_prefix(self, prefix):
#Step 1: Traverse to the end of the prefix
    node = self.root
    for char in prefix:
    node = self._find_child(node, char)
    if not node:
        return None
# Step 2: BFS
     queue = Queue()
     queue.enqueue((node, prefix))
     while not queue.is_empty():
        current_node, path = queue.dequeue()
        if current_node.is_end_of_word:
            return path
        child = current_node.first_child
        while child:
            queue.enqueue((child,path+child.char))
            child = child.next_sibling
        return None
```




```
def find_shortest_word_with_prefix(self, prefix):
#Step 1: Traverse to the end of the prefix
    node = self.root
    for char in prefix:
    node = self._find_child(node, char)
    if not node:
        return None
# Step 2: BFS
     queue = Queue()
     queue.enqueue((node, prefix))
     while not queue.is_empty():
        current_node, path = queue.dequeue()
        if current_node.is_end_of_word:
            return path
        child = current_node.first_child
        while child:
            queue.enqueue((child,path+child.char))
            child = child.next_sibling
        return None
```




```python
queue.enqueue((node, prefix, 0))  # (node, word, depth)
shortest_words = []
min_depth = None

while not queue.is_empty():
    node, word, depth = queue.dequeue()

    if node.is_end_of_word:
        if min_depth is None:
            min_depth = depth
        if depth == min_depth:
            shortest_words.append(word)
        elif depth > min_depth:
            break  # We already found the shortest level, skip deeper

    if min_depth is not None and depth >= min_depth:
        continue  # don't enqueue deeper nodes

    child = node.first_child
    while child:
        queue.enqueue((child, word + child.char, depth + 1))
        child = child.next_sibling

return shortest_words
```