String Data Structures

what is the cutest ani

what is the cutest animal in the world
what is the cutest animal
what is the cutest animal on earth
what is the cutest animal ever
what is the cutest animal in the whole entire world
what is the cutest animal in the whole world
what is the cutest animal alive
what is the cutest animal on the planet
what is the cutest animal in australia
what is the cutest animal in the sea

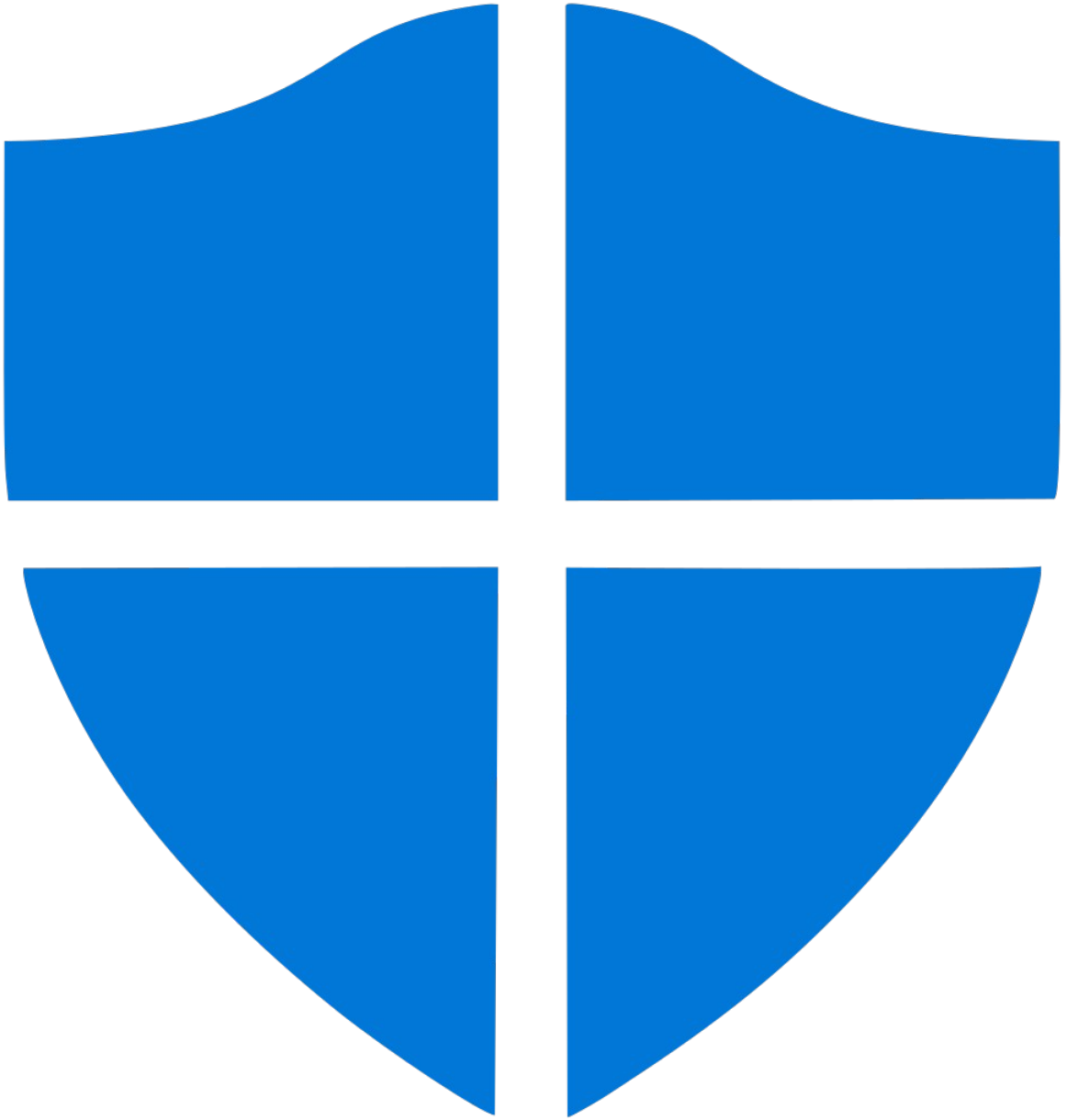
Google Search | I'm Feeling Lucky

Report inappropriate predictions

Why do we need string data structures?

Why do we need string data structures?

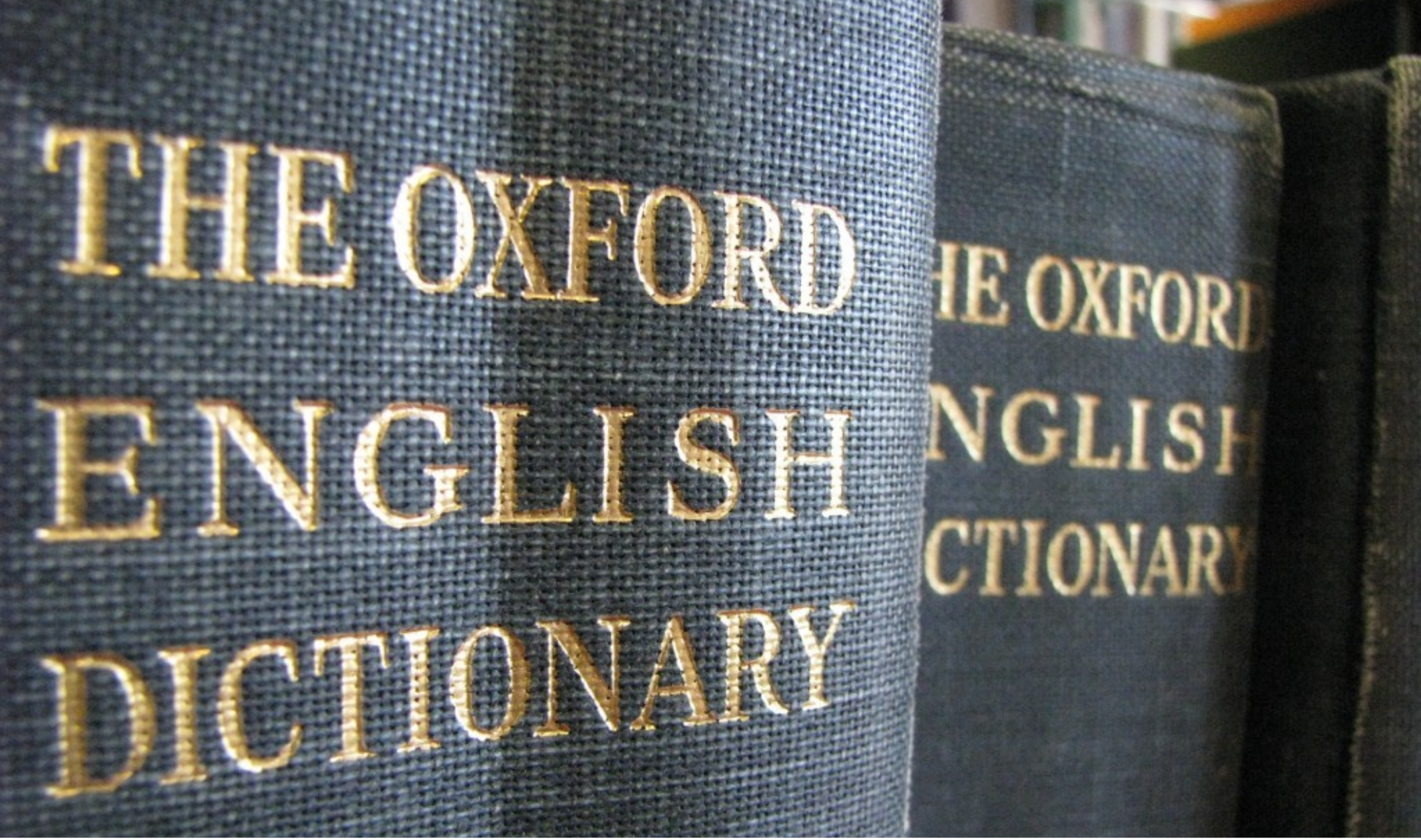
Why do we need string data structures?



Why do we need string data structures?

String Data Structures

Before Computers



Can binary search for a word in time O(log n).
Can ***interpolation search*** in average time O(log log n).

Array accesses take time O(1).

Jump to the drawer, then do an O(d) lookup, where d is the number of elements in the drawer.

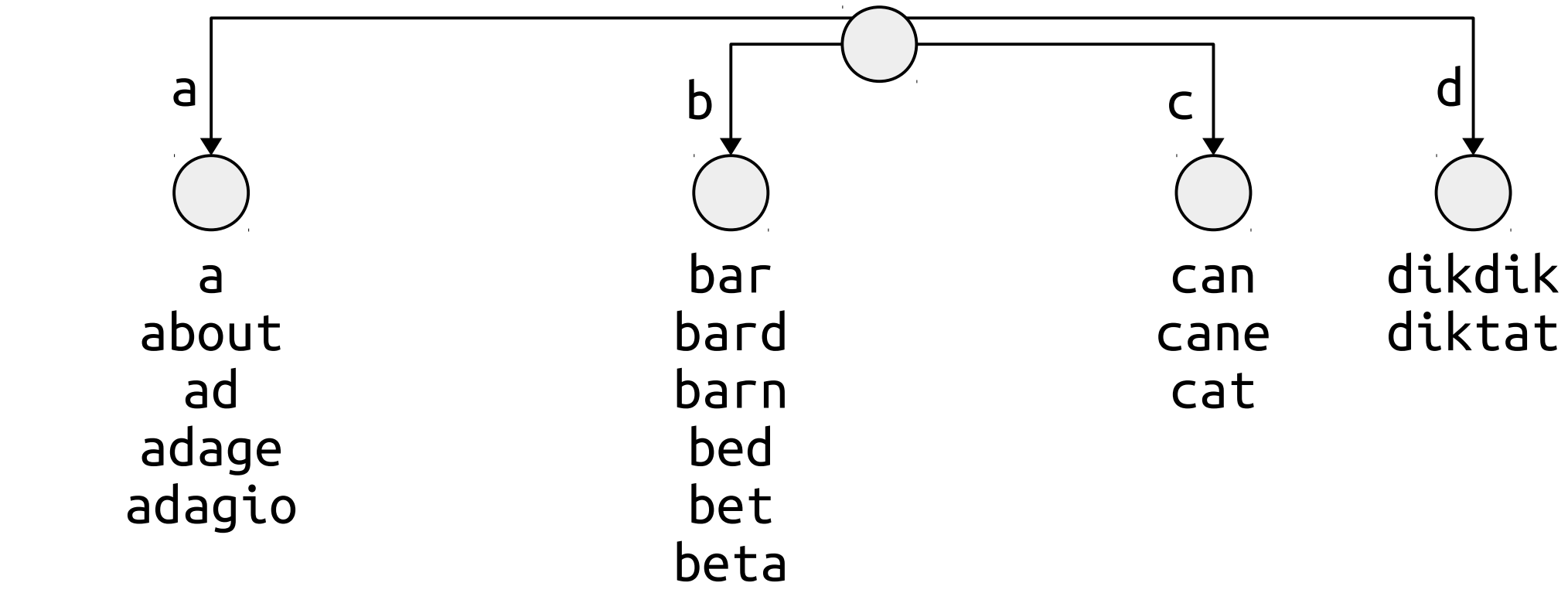
Miriam-Webster's physical "Backward Index:"
All English words, written in reverse, in sorted order.
Why would you want this?

Find all words ending in “iatrics.”
Time required: **O(log *n* + *k*)**.

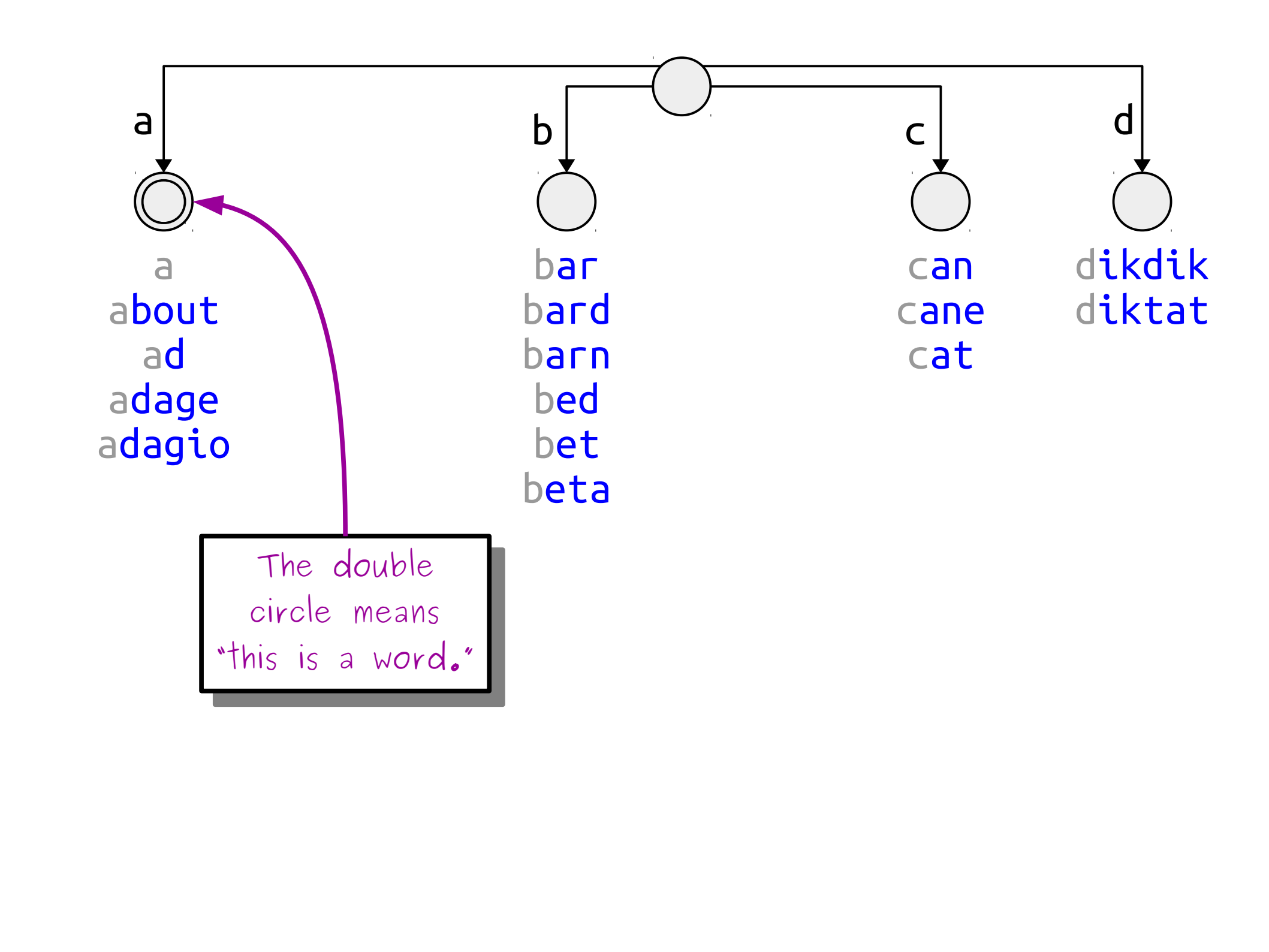
String Data Structures

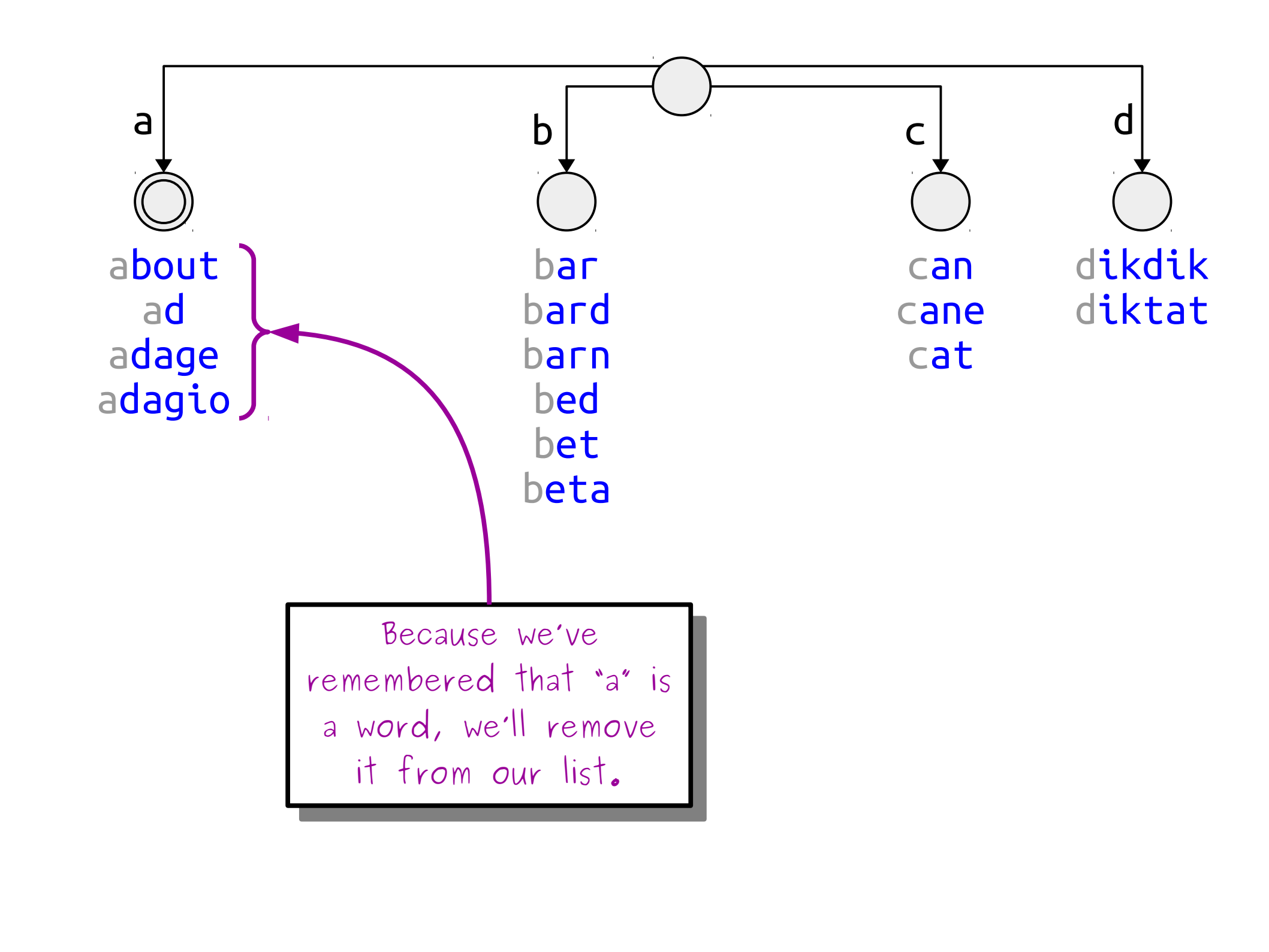
With Computers

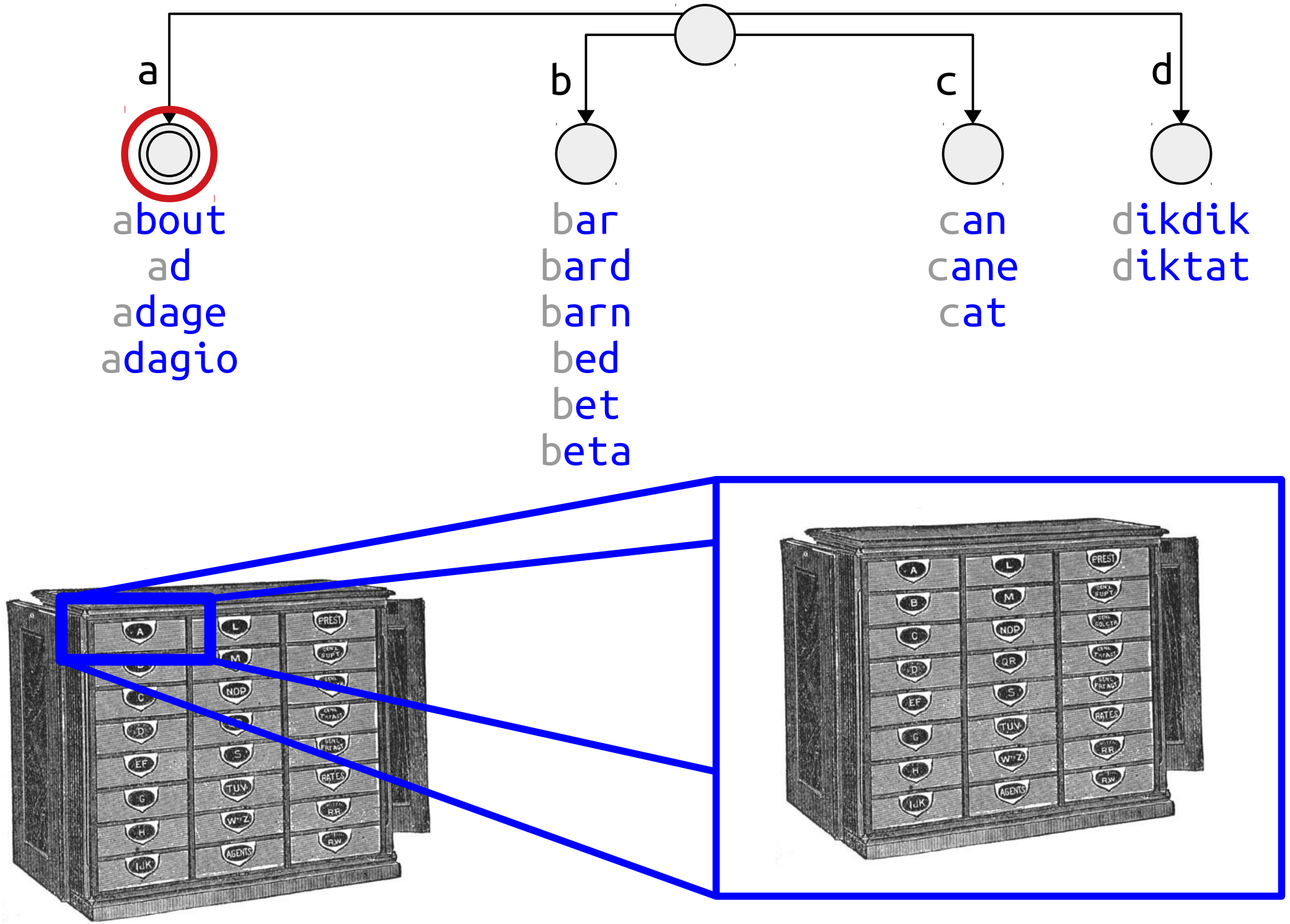
a
about
ad
adage
adagio
bar
bard
barn
bed
bet
beta
can
cane
cat
dikdik
diktat

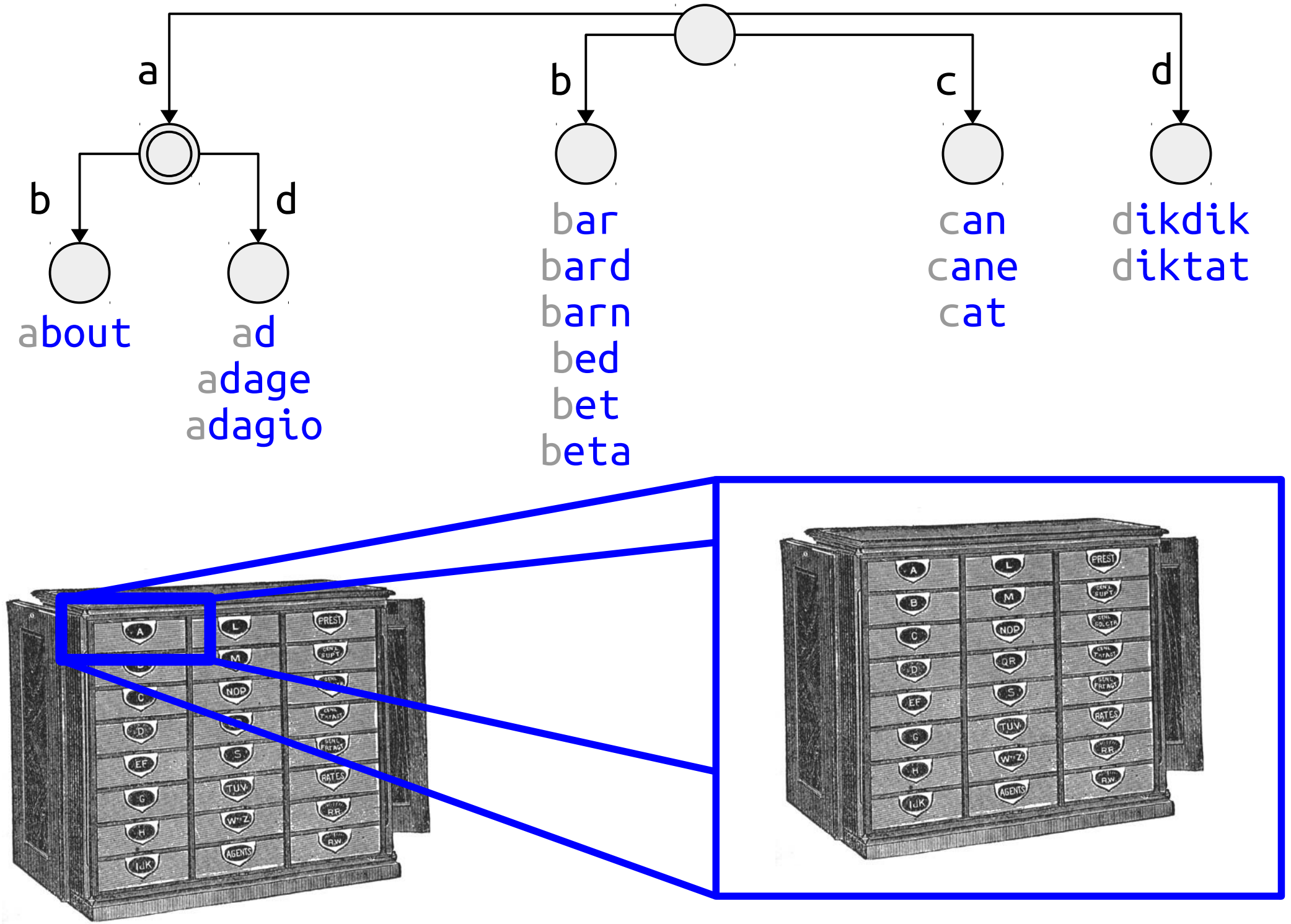

a
b
c
d
a
about
ad
adage
adagio
bar
bard
barn
bed
bet
beta
can
cane
cat
dikdik
diktat

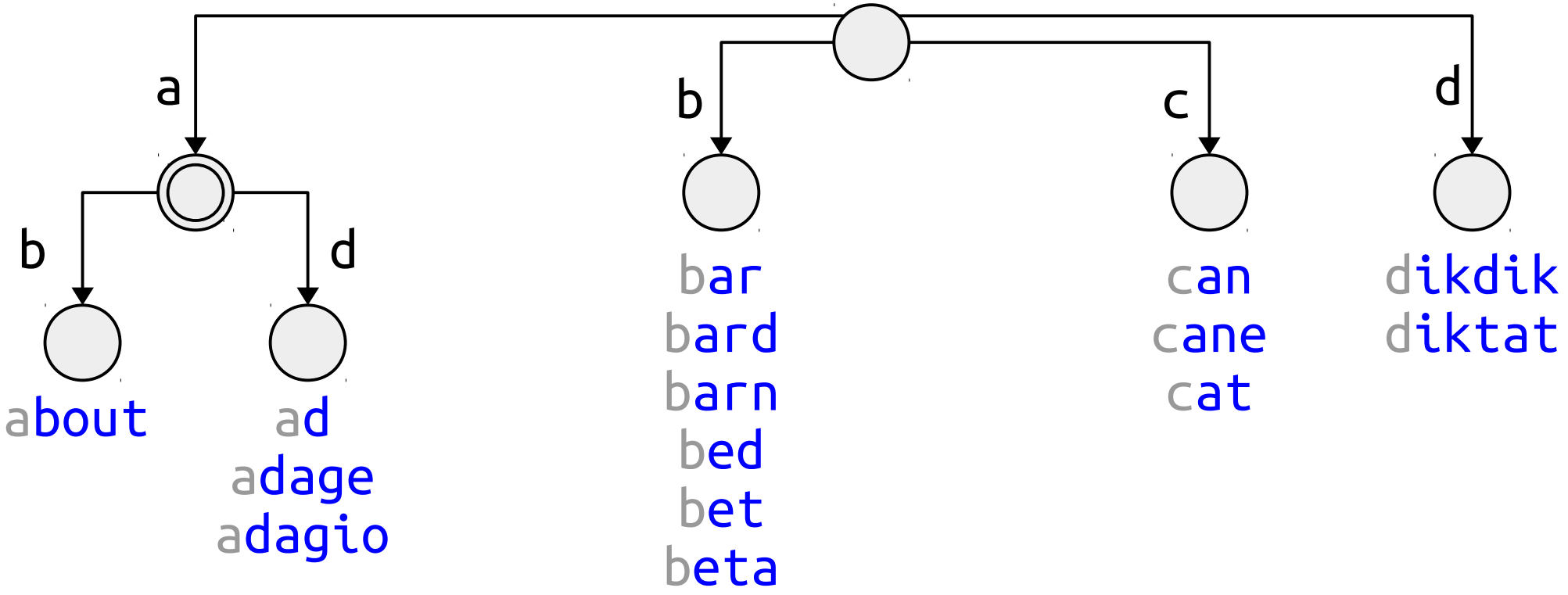

A
B
C
D
EF
G
H
IJK
L
M
NOP
QR
S
TUV
W-Z
AGENTS
PREST
RATES
R.W

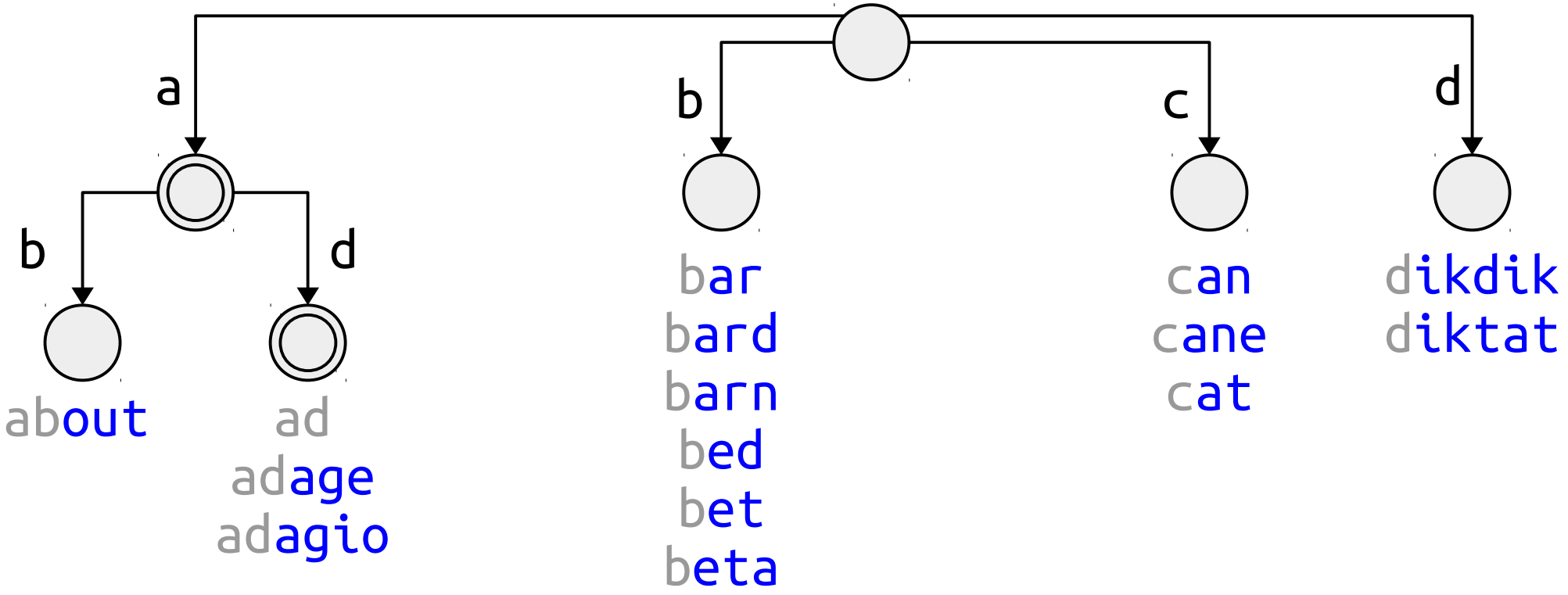

a
b
c
d
a
about
ad
adage
adagio
bar
bard
barn
bed
bet
beta
can
cane
cat
dikdik
diktat
The double circle means "this is a word."

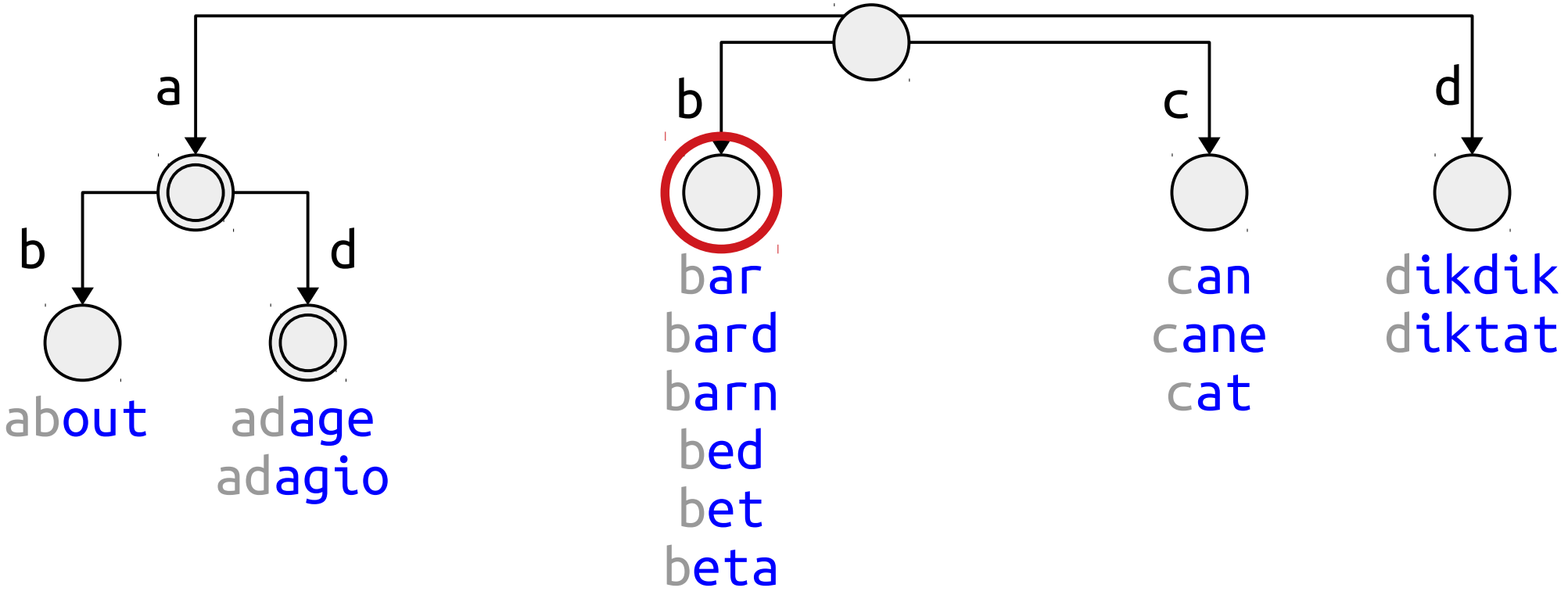

a
b
c
d
about
ad
adage
adagio
bar
bard
barn
bed
bet
beta
can
cane
cat
dikdik
diktat
Because we've remembered that "a" is a word, we'll remove it from our list.

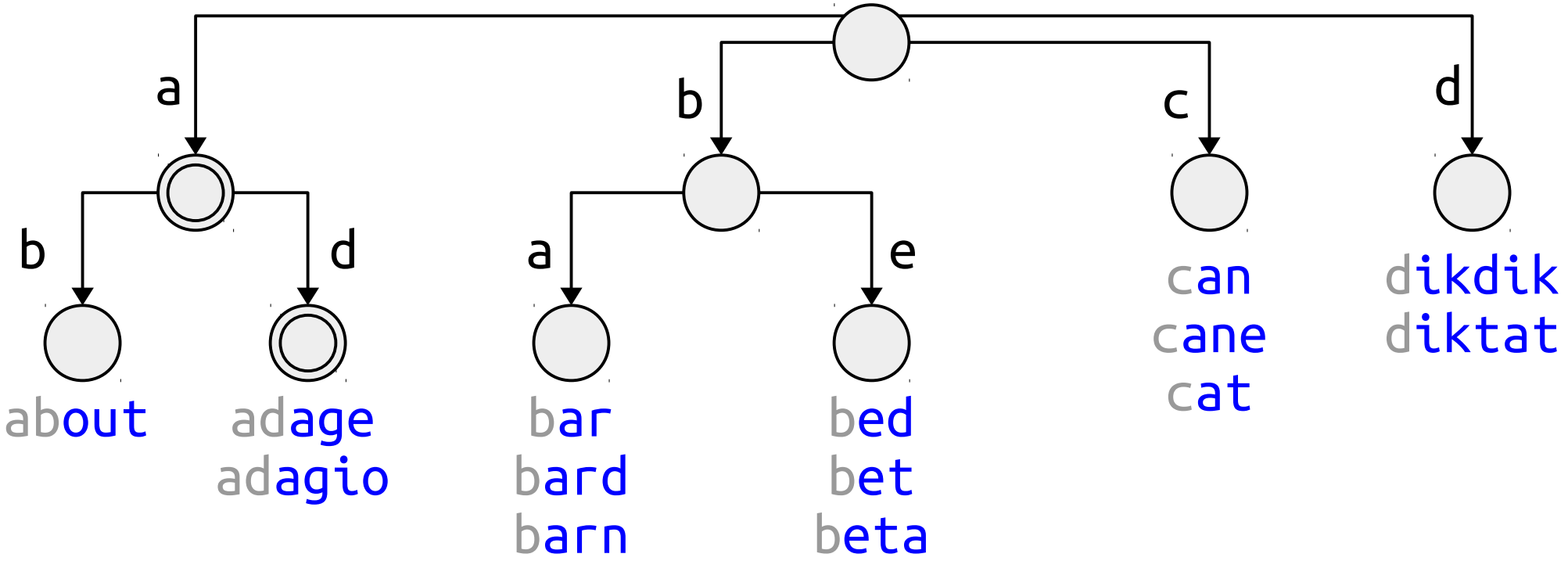

a
b
c
d
about
ad
adage
adagio
bar
bard
barn
bed
bet
beta
can
cane
cat
dikdik
diktat

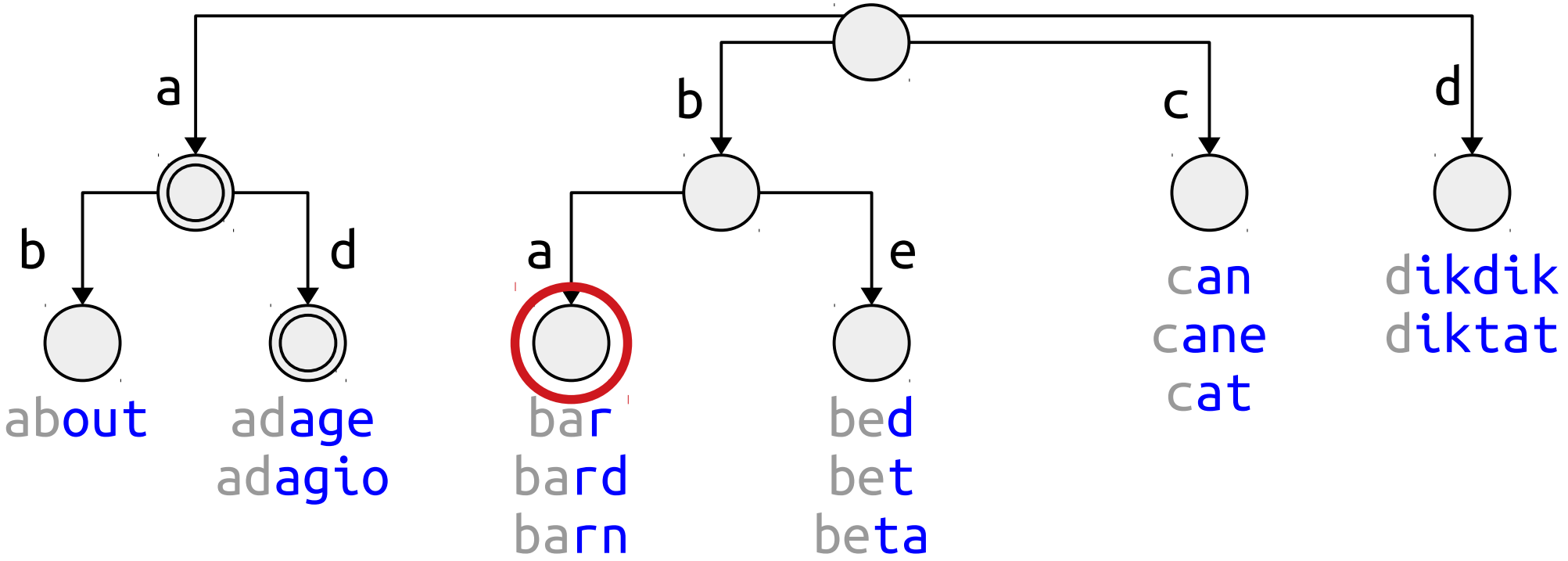

a
b
d
about
ad
adage
adagio
b
c
d
bar
bard
barn
bed
bet
beta
can
cane
cat
dikdik
diktat

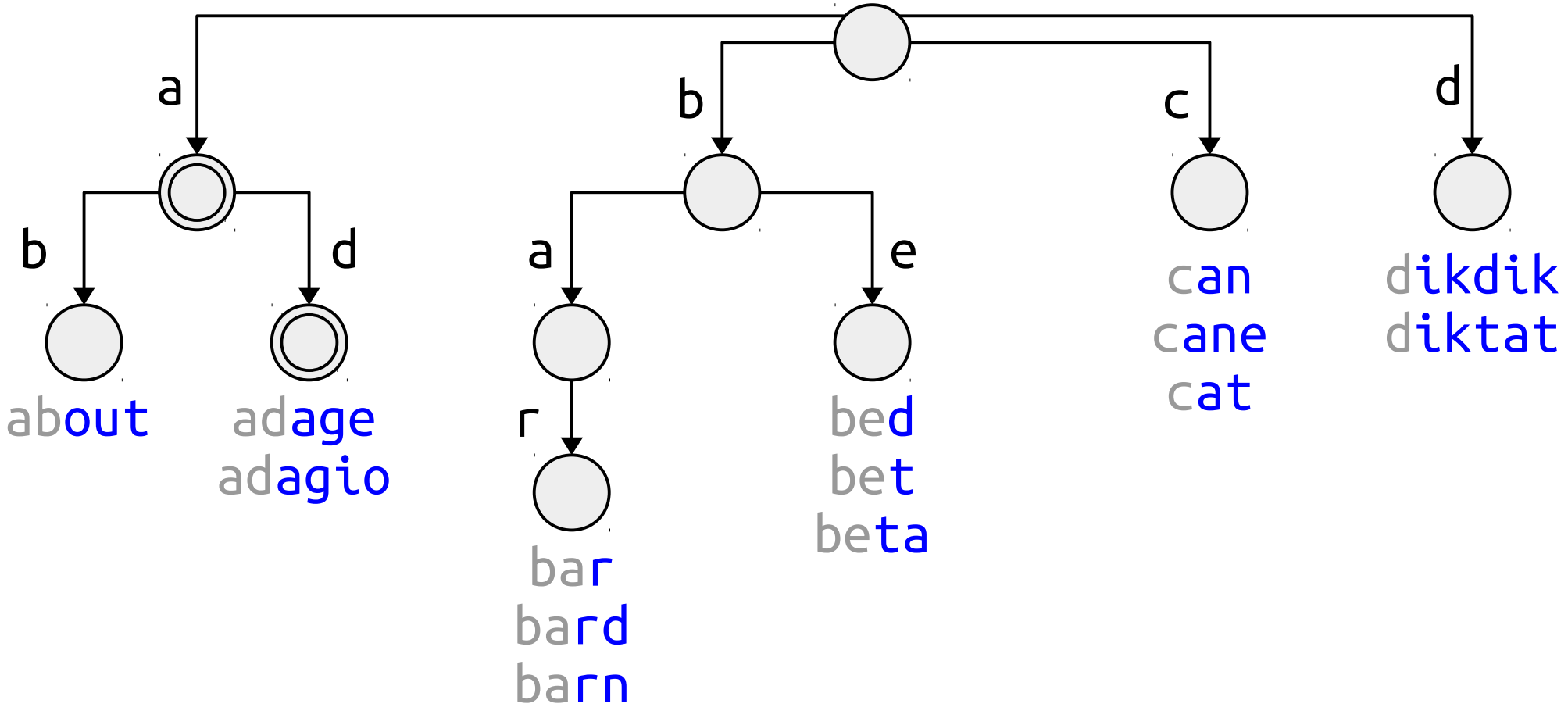

a
b
d
about
ad
adage
adagio
b
bar
bard
barn
bed
bet
beta
c
can
cane
cat
d
dikdik
diktat

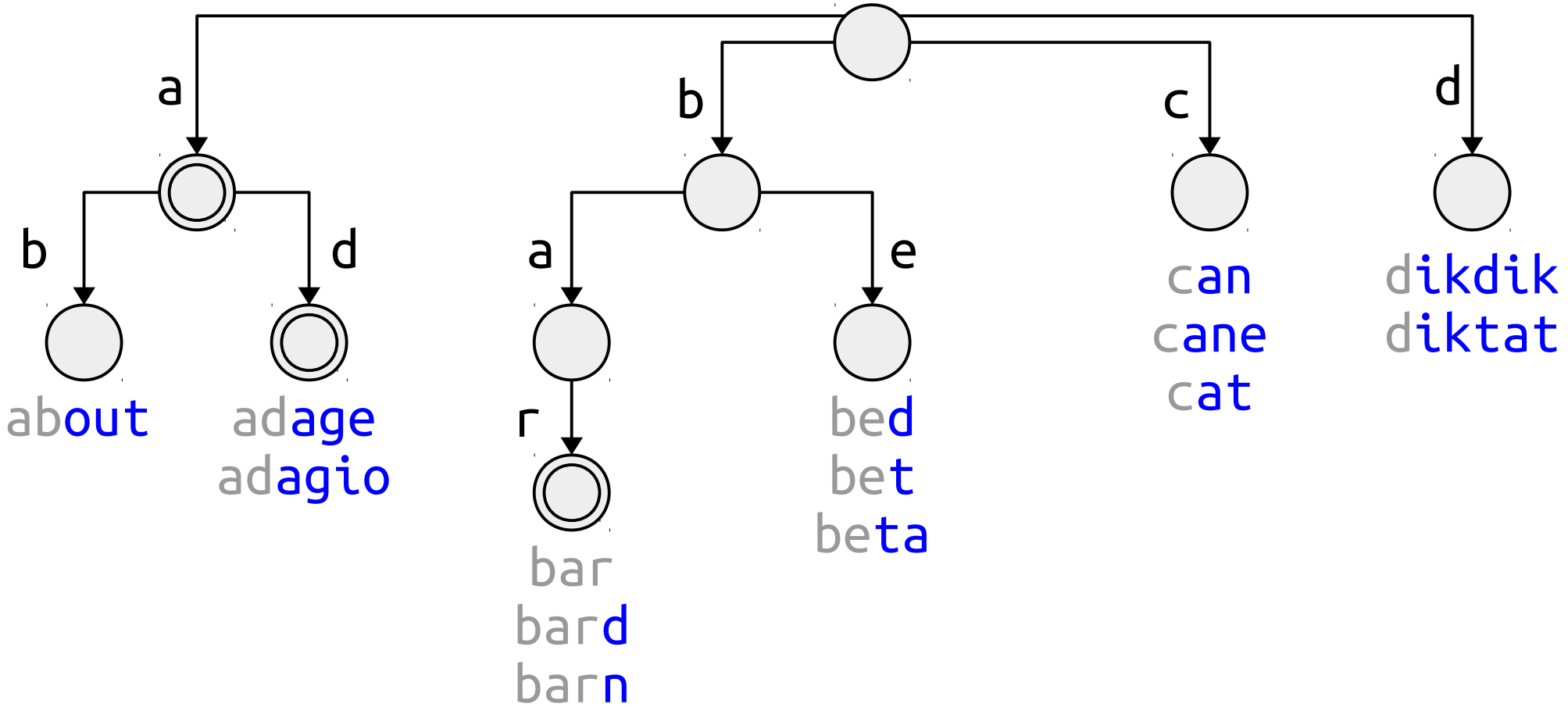

a
b
d
about
ad
adage
adagio
b
bar
bard
barn
bed
bet
beta
c
can
cane
cat
d
dikdik
diktat

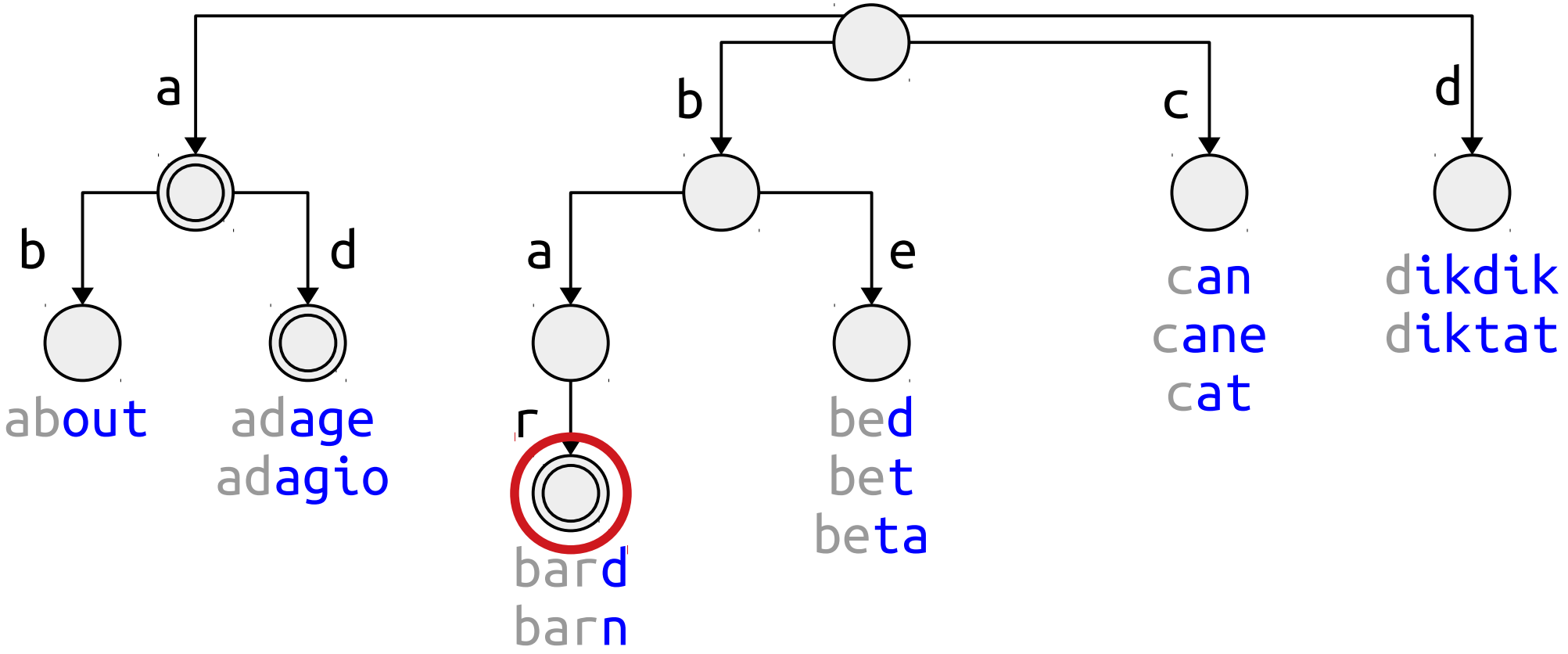

a
b
d
about
adage
adagio
b
bar
bard
barn
bed
bet
beta
c
can
cane
cat
d
dikdik
diktat

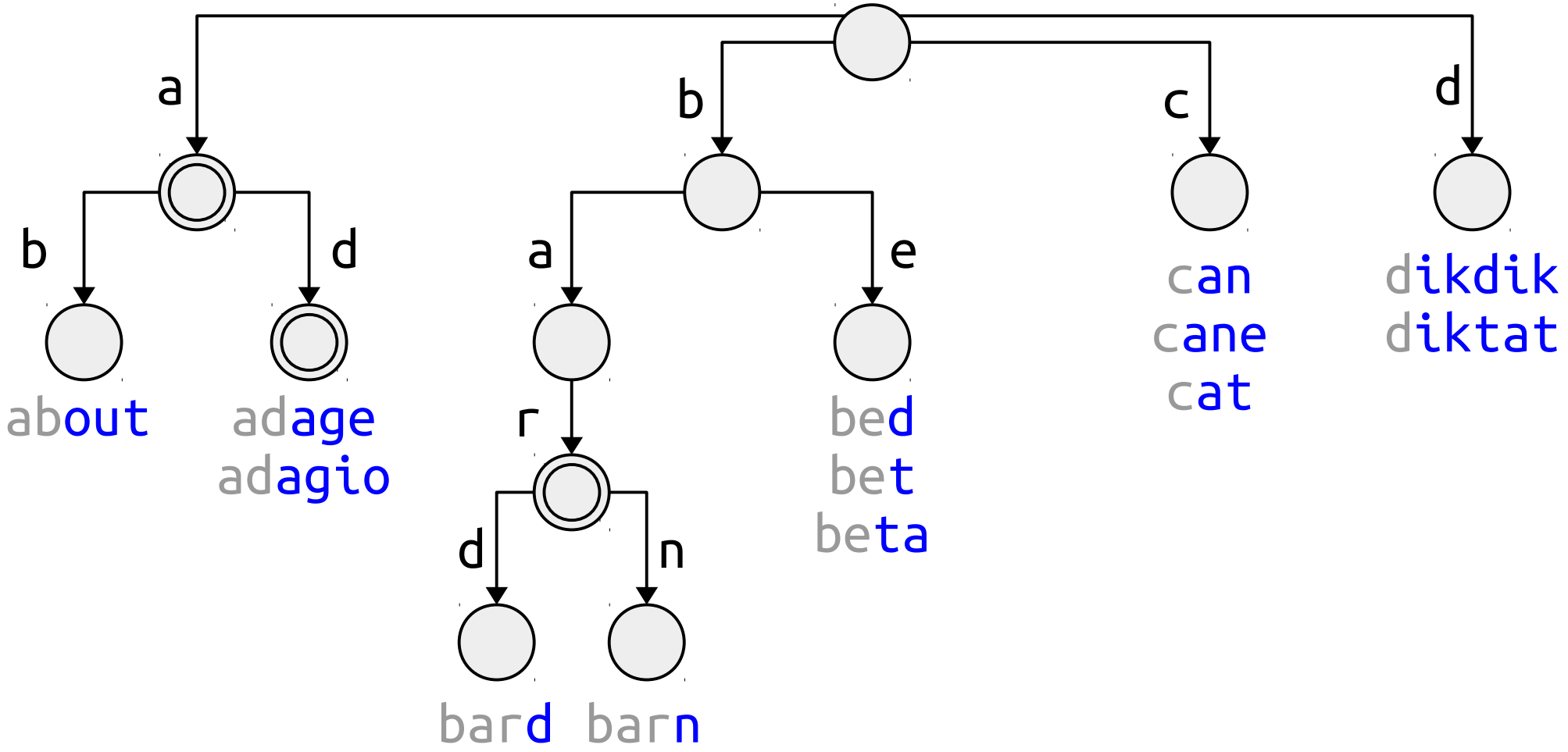

a
b
c
d
b
d
a
e
about
adage
adagio
bar
bard
barn
bed
bet
beta
can
cane
cat
dikdik
diktat

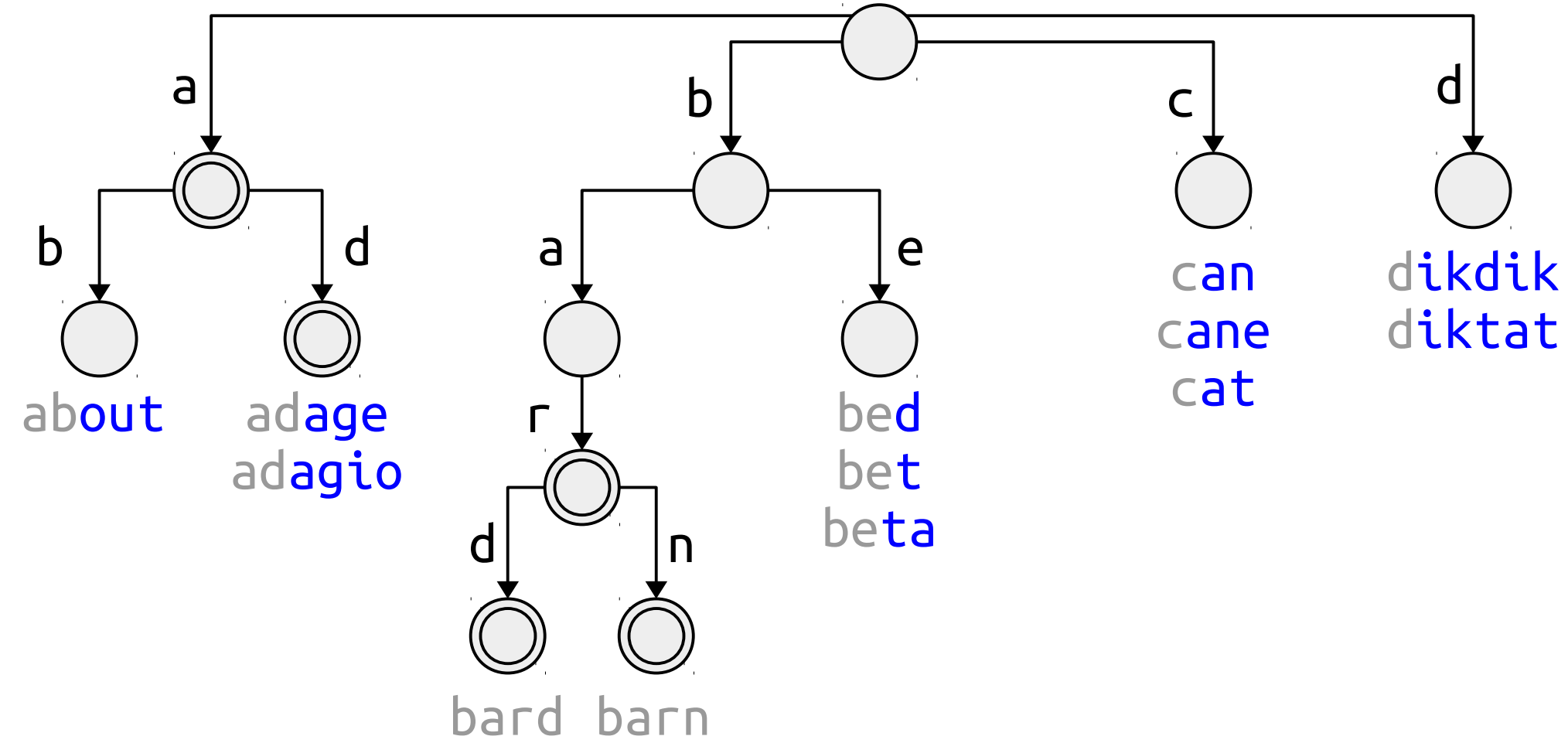

a
b
c
d
b
d
a
e
about
adage
adagio
bar
bard
barn
bed
bet
beta
can
cane
cat
dikdik
diktat

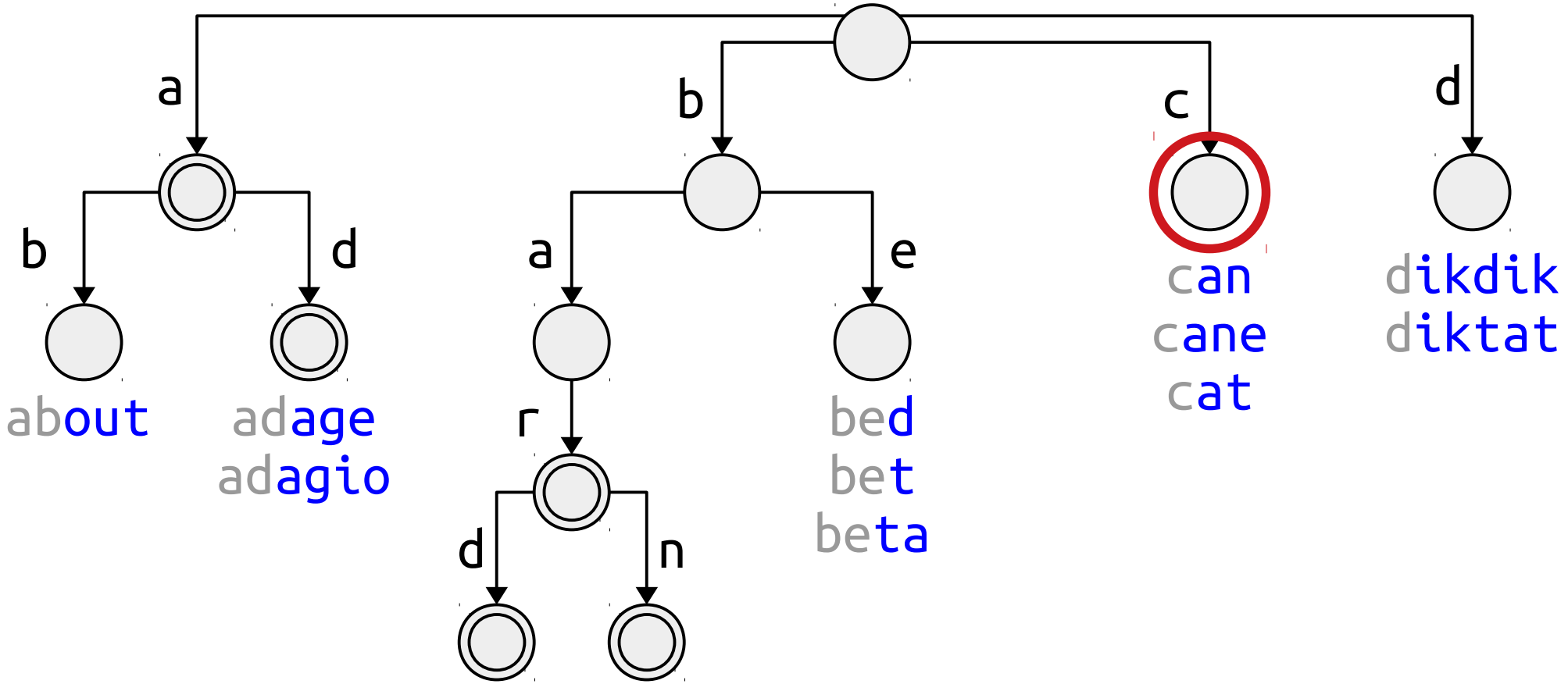

a
b
c
d
b
d
a
e
r
about
adage
adagio
bar
bard
barn
bed
bet
beta
can
cane
cat
dikdik
diktat

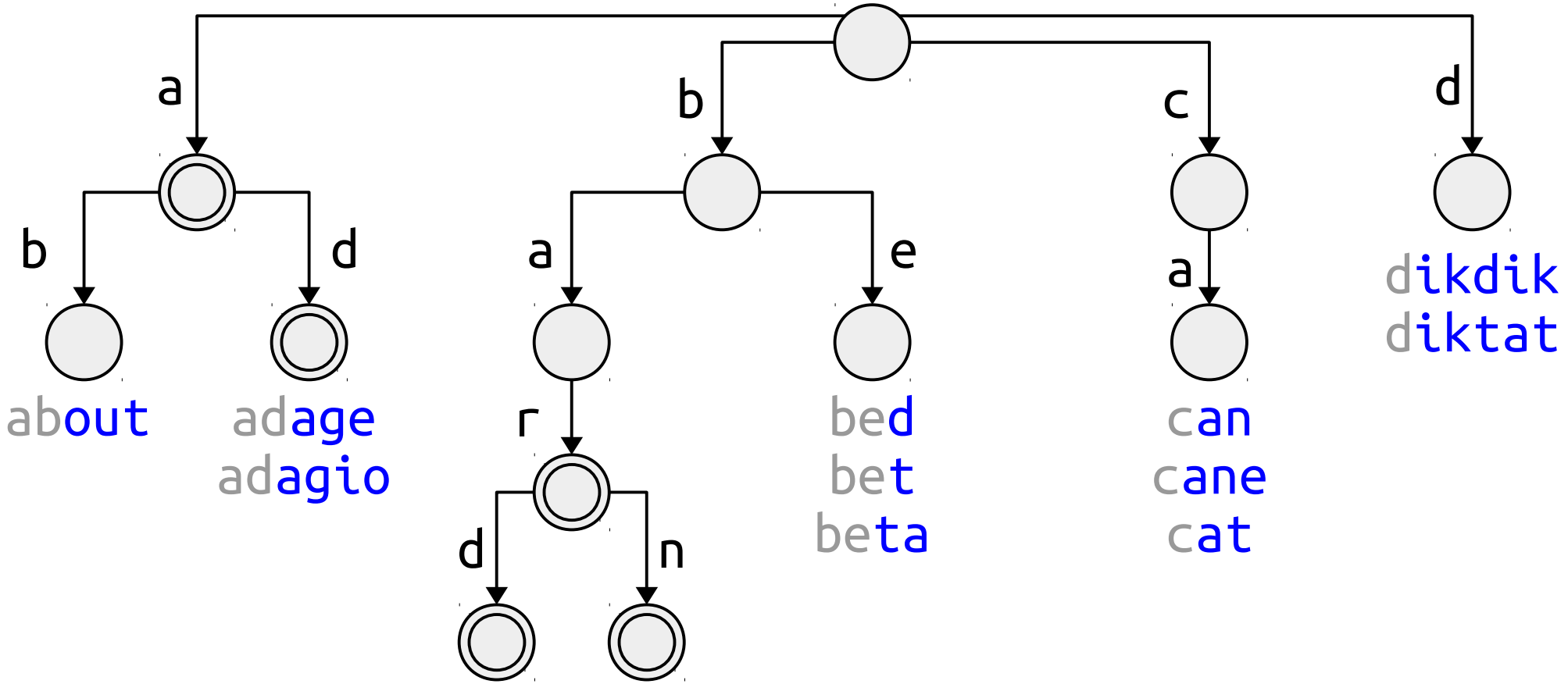

a
b
c
d
b
d
a
e
about
adage
adagio
r
bed
bet
beta
can
cane
cat
dikdik
diktat
bar
bard
barn

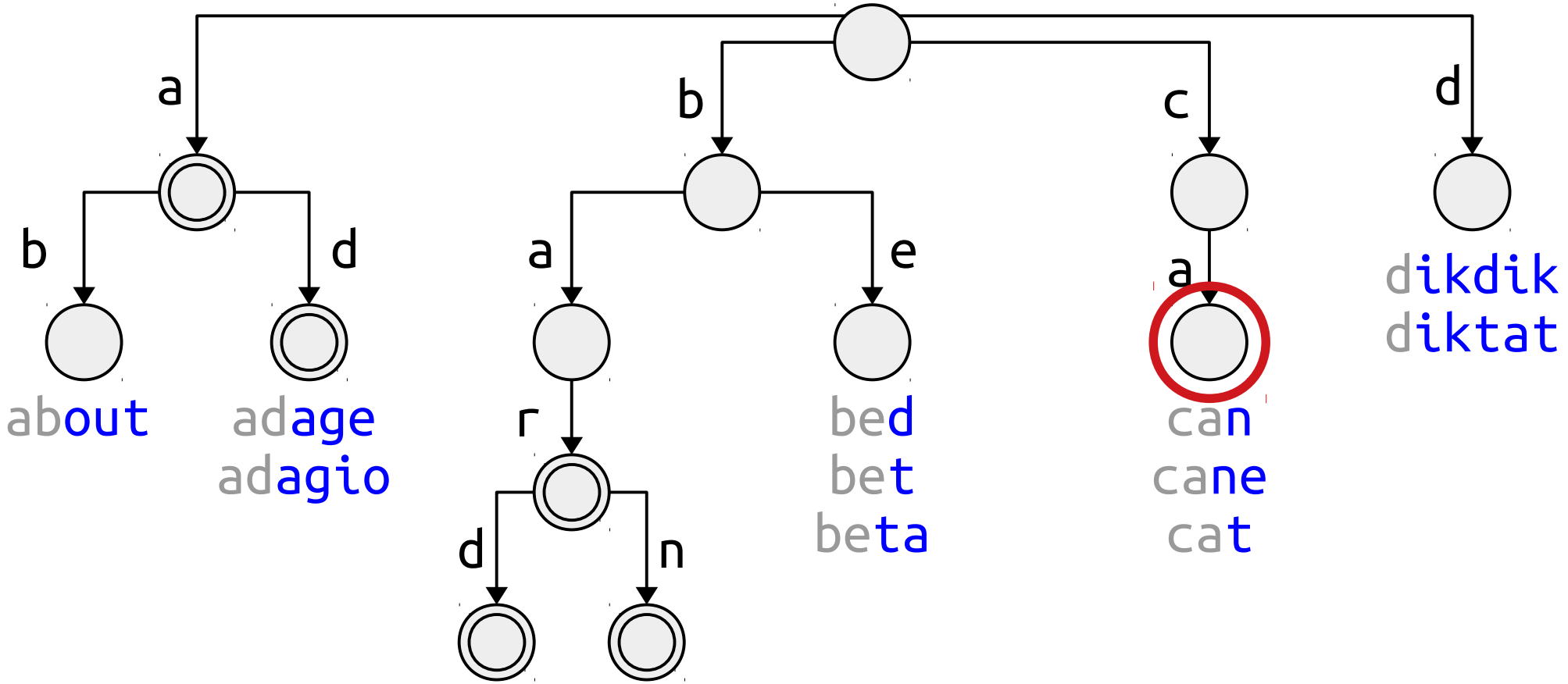

a
b
c
d
b
d
a
e
about
adage
adagio
r
bed
bet
beta
can
cane
cat
dikdik
diktat
bard
barn

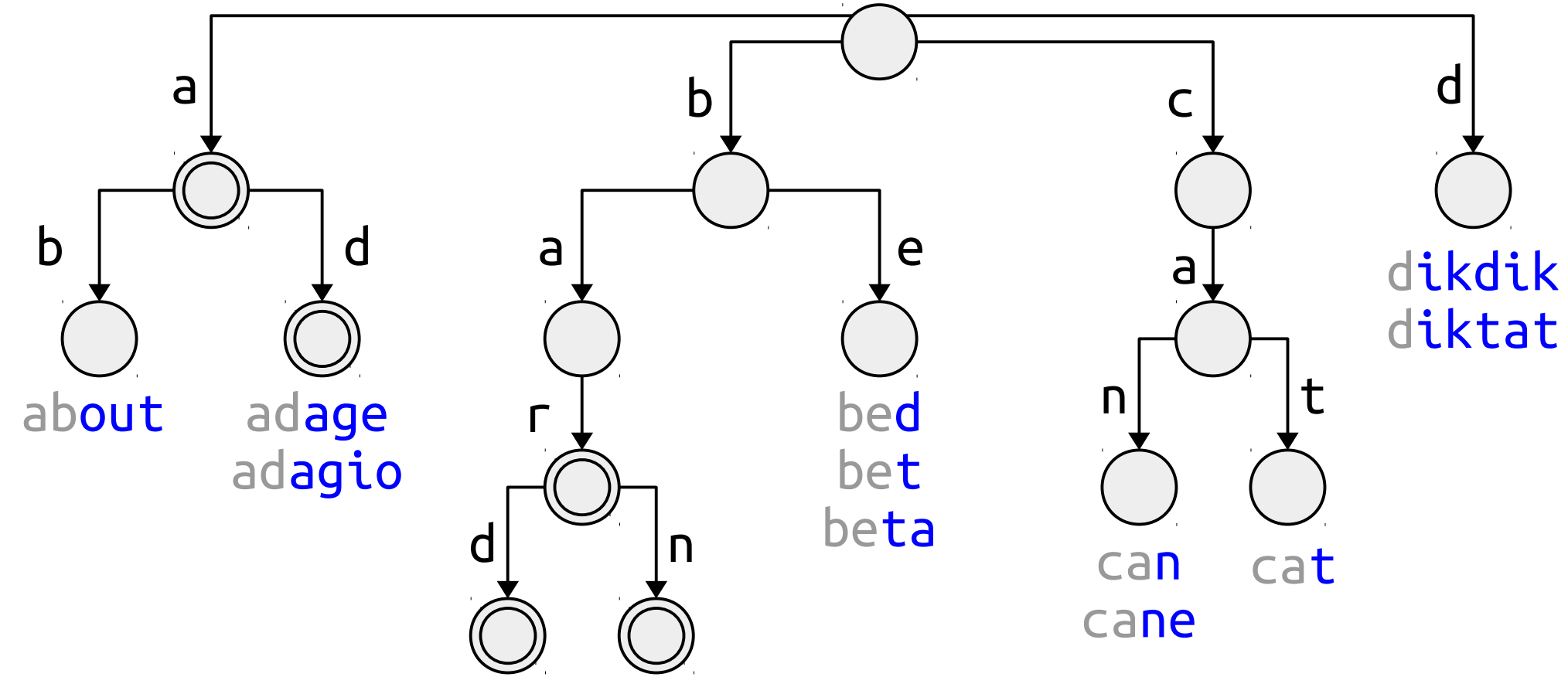

a
b
c
d
b
d
a
e
r
d
n
about
adage
adagio
bard
barn
bed
bet
beta
can
cane
cat
dikdik
diktat

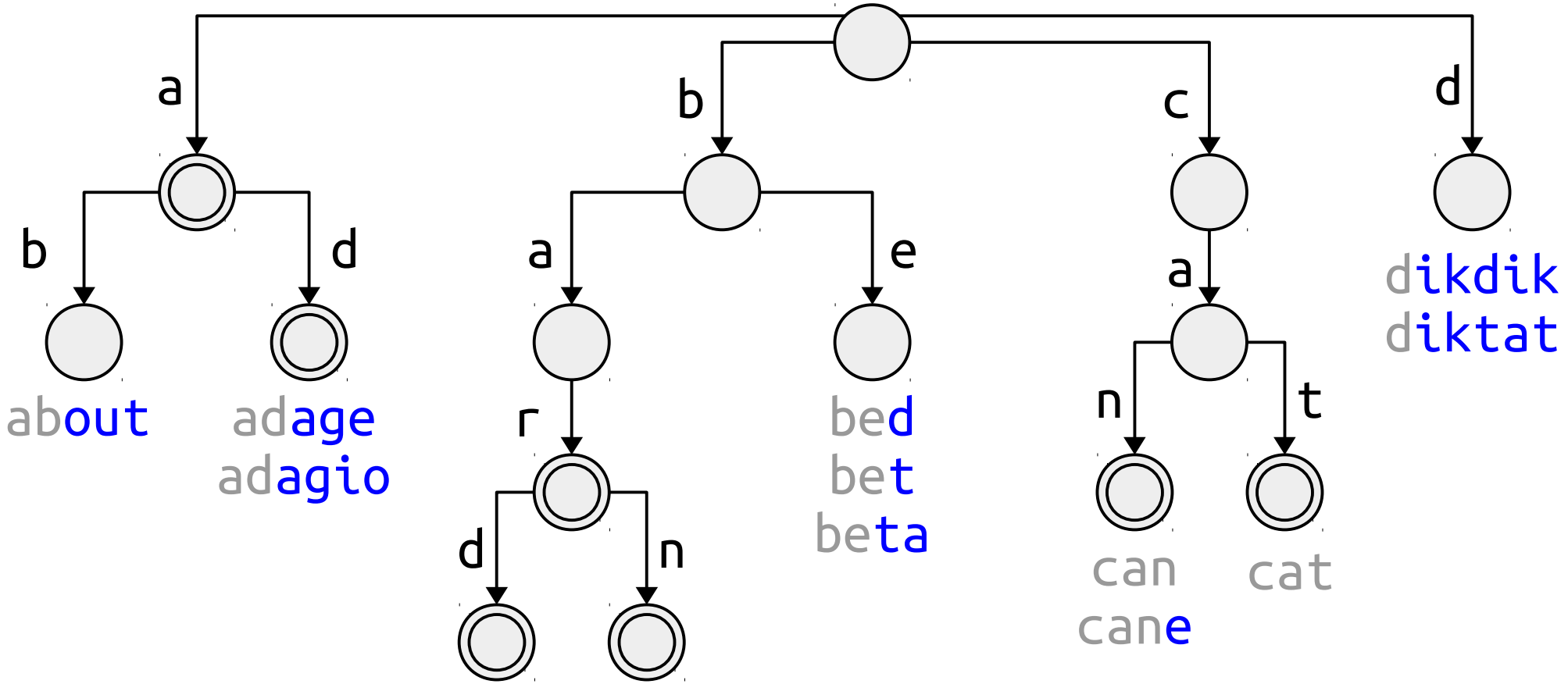

a
b
c
d
b
d
a
e
about
adage
adagio
r
bed
bet
beta
can
cane
cat
dikdik
diktat
d
n
bard
barn

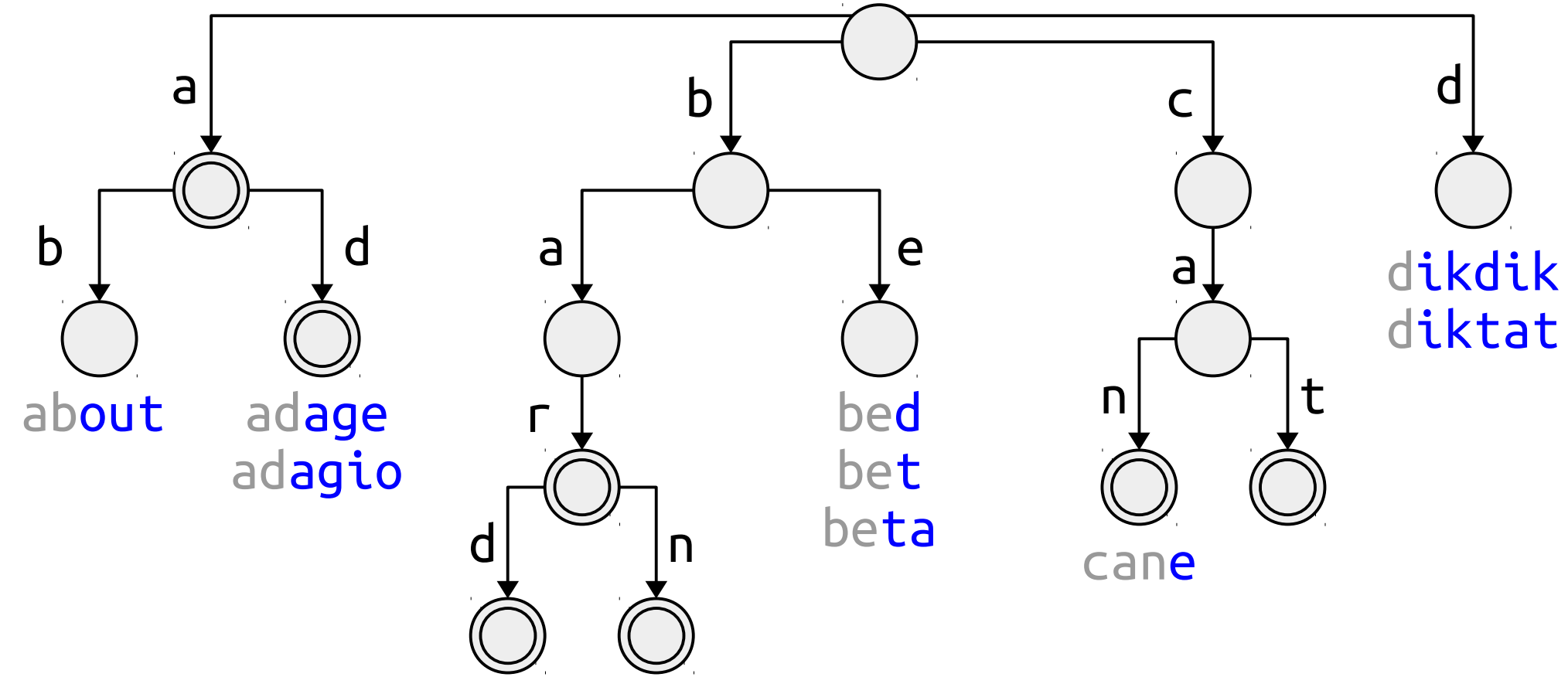

a
b
c
d
b
d
a
e
about
adage
adagio
r
bed
bet
beta
can
cane
cat
dikdik
diktat
d
n

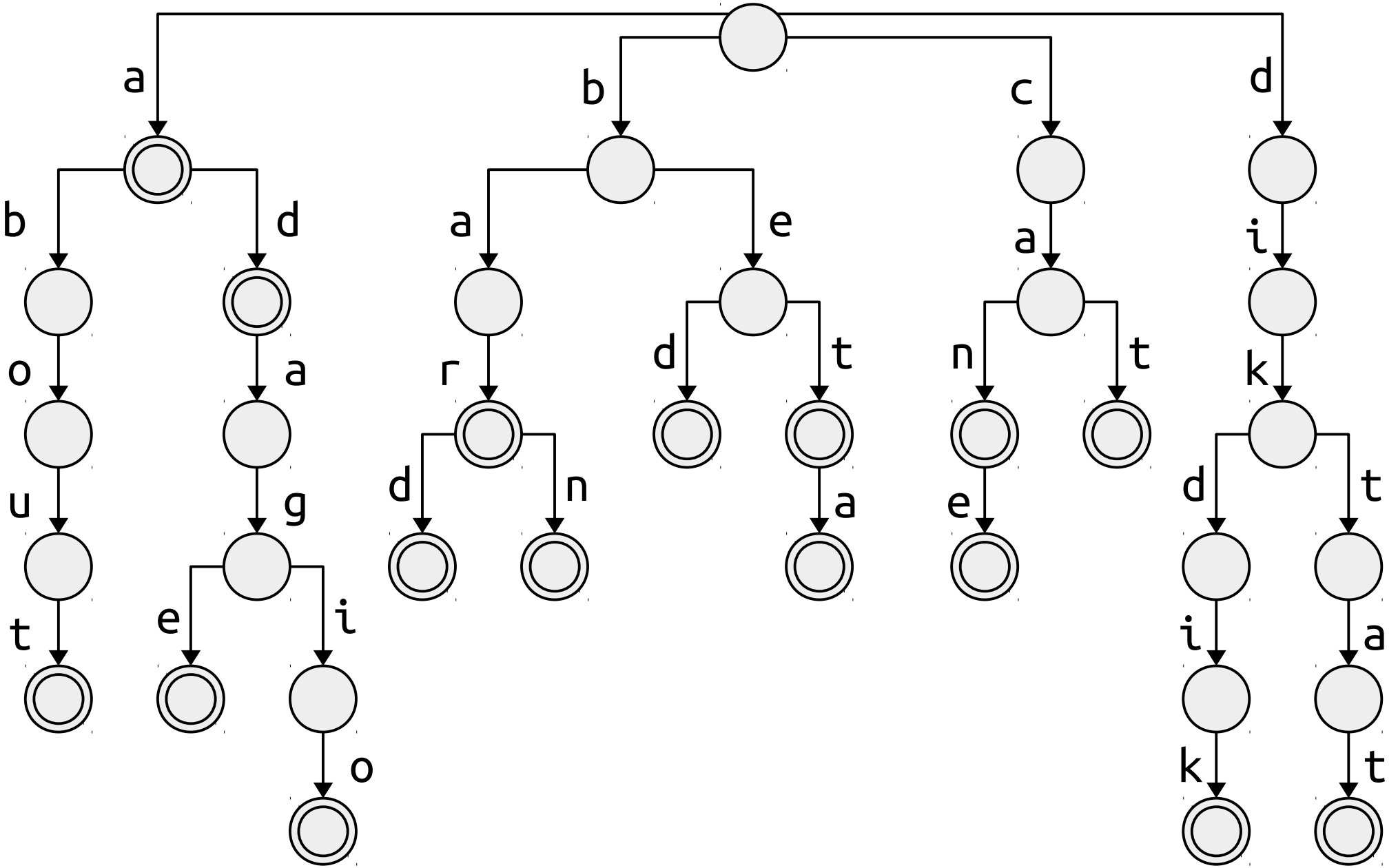

a
b
d
about
adage
adagio
b
a
e
r
d
n
bed
bet
beta
c
a
can
cane
cat
d
dikdik
diktat

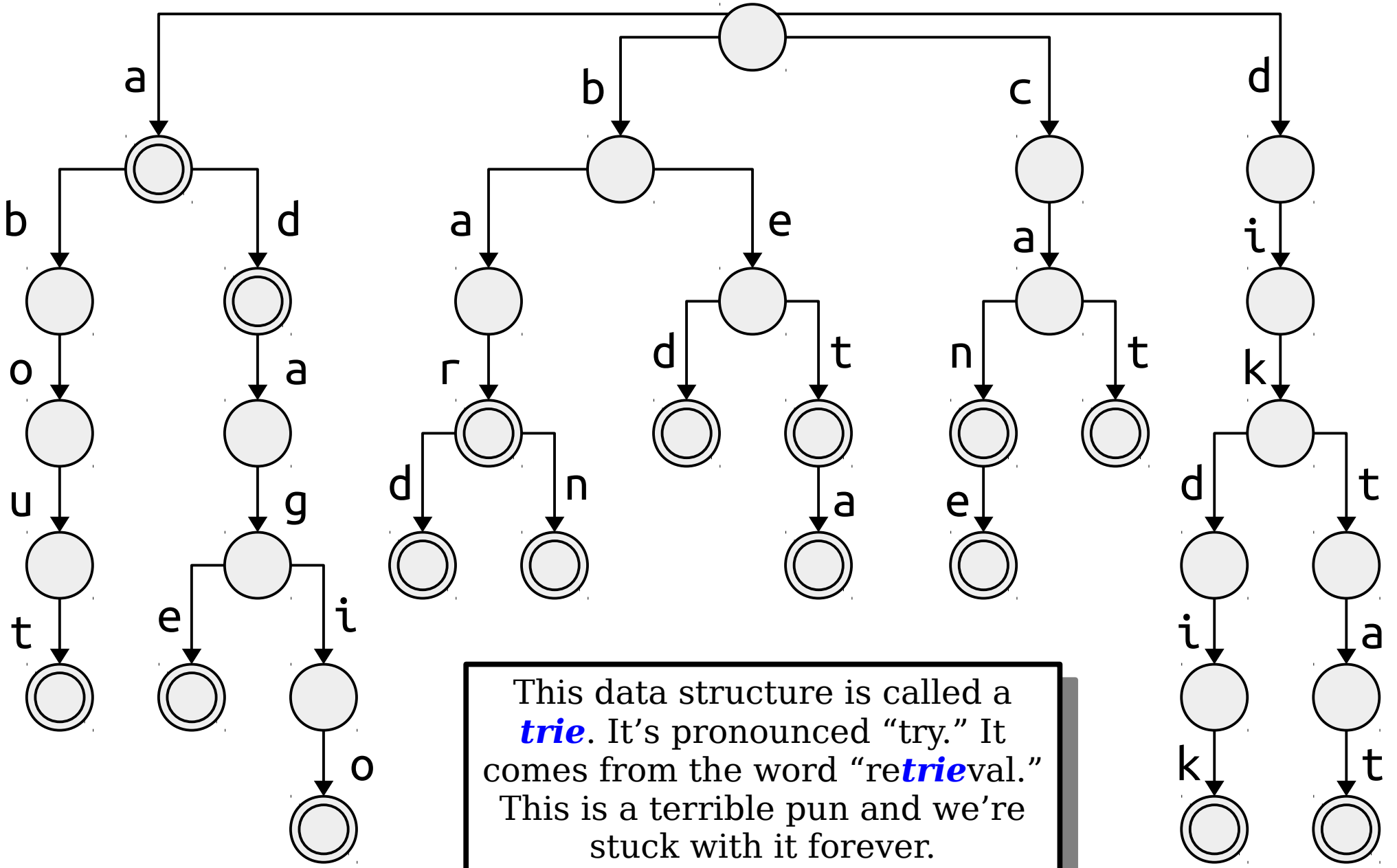

a
b
d
about
adage
adagio
b
a
e
r
d
n
bed
bet
beta
c
a
can
cane
cat
d
dikdik
diktat


a
b
d
about
adage
adagio
b
a
e
r
d
n
bed
bet
beta
c
a
n
t
can
cane
cat
d
dikdik
diktat


a
b
d
about
adage
adagio
b
a
e
r
d
n
bed
bet
beta
c
a
n
t
can
cane
cat
d
dikdik
diktat

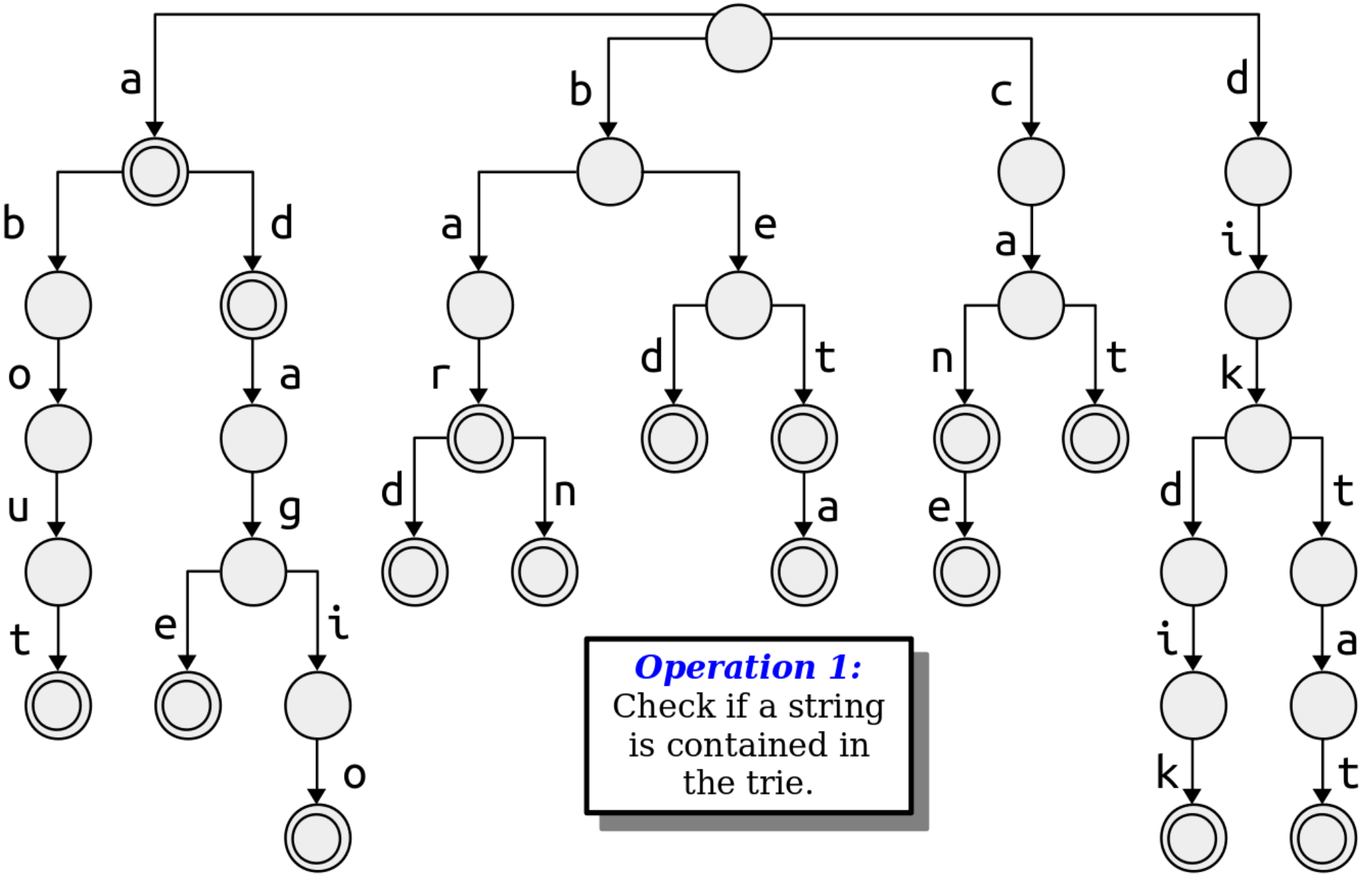

a
b
d
about
adage
adagio
b
a
e
r
d
n
bed
bet
beta
c
a
n
t
cane
d
dikdik
diktat


a
b
c
d
b
d
a
e
a
i
o
a
r
d
t
n
t
k
u
g
d
n
d
a
e
d
t
t
e
i
i
a
o
k
t

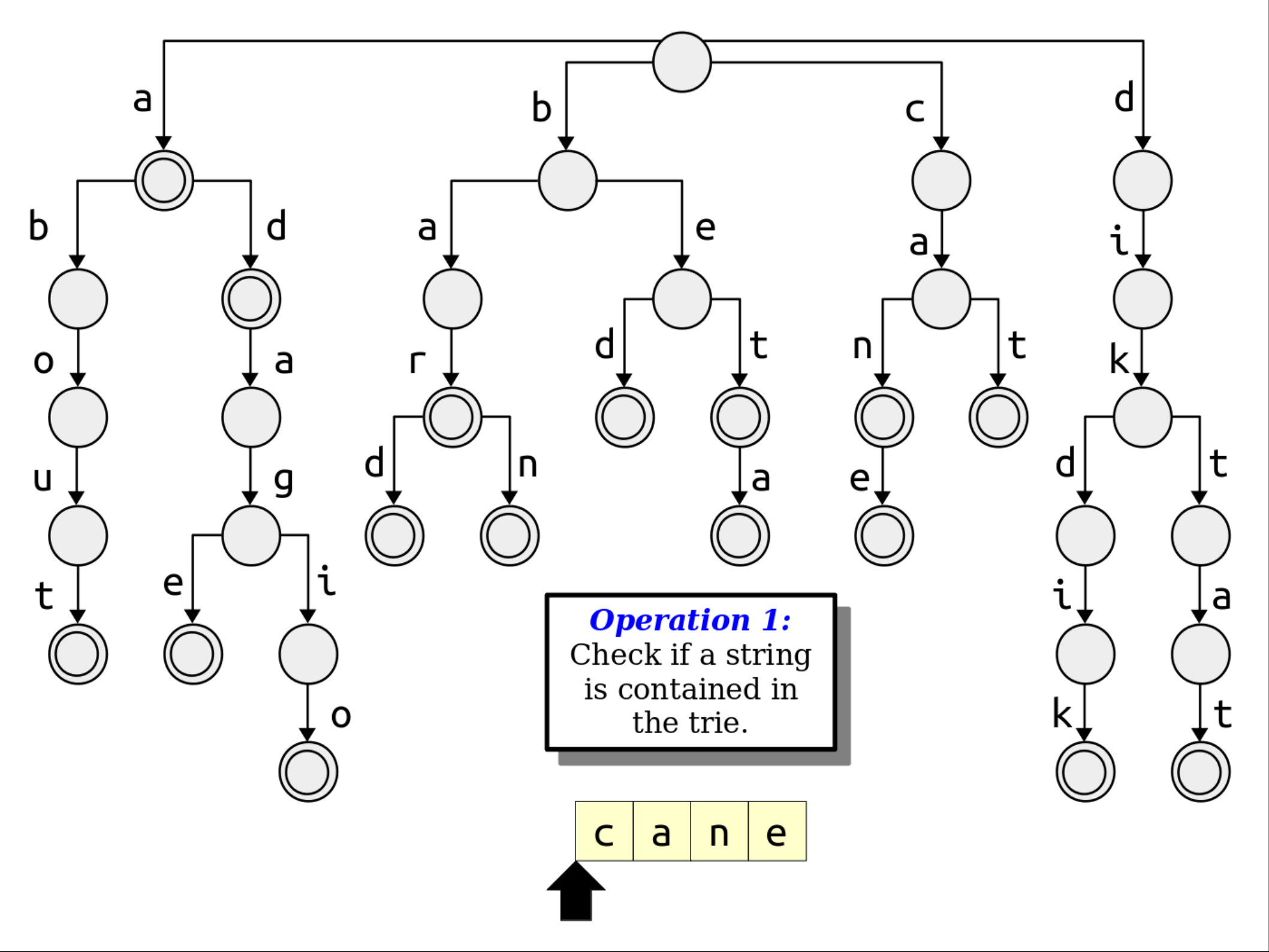


This data structure is called a ***trie***. It's pronounced "try." It comes from the word "re***trie***val." This is a terrible pun and we're stuck with it forever.

Using a Trie

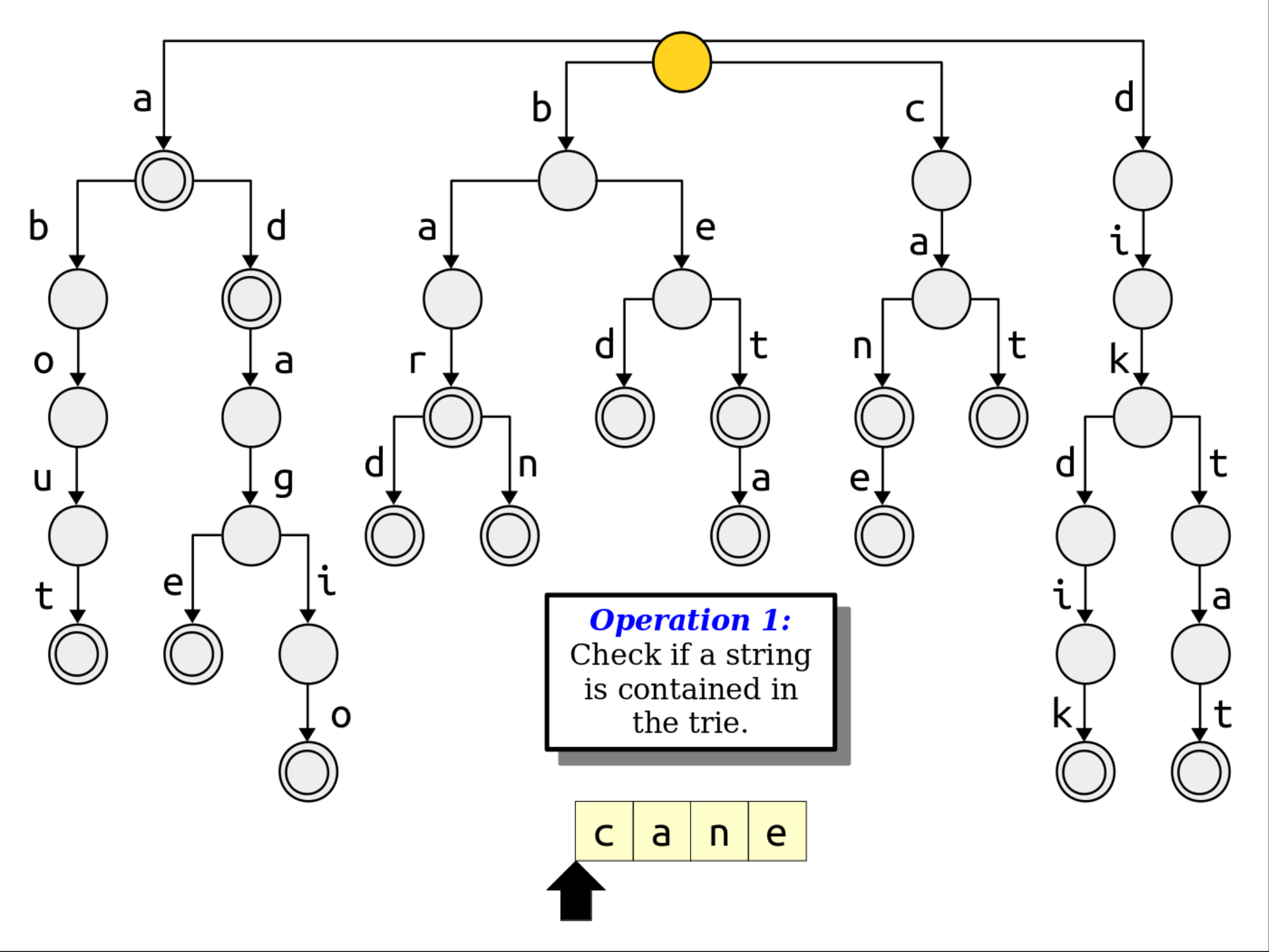


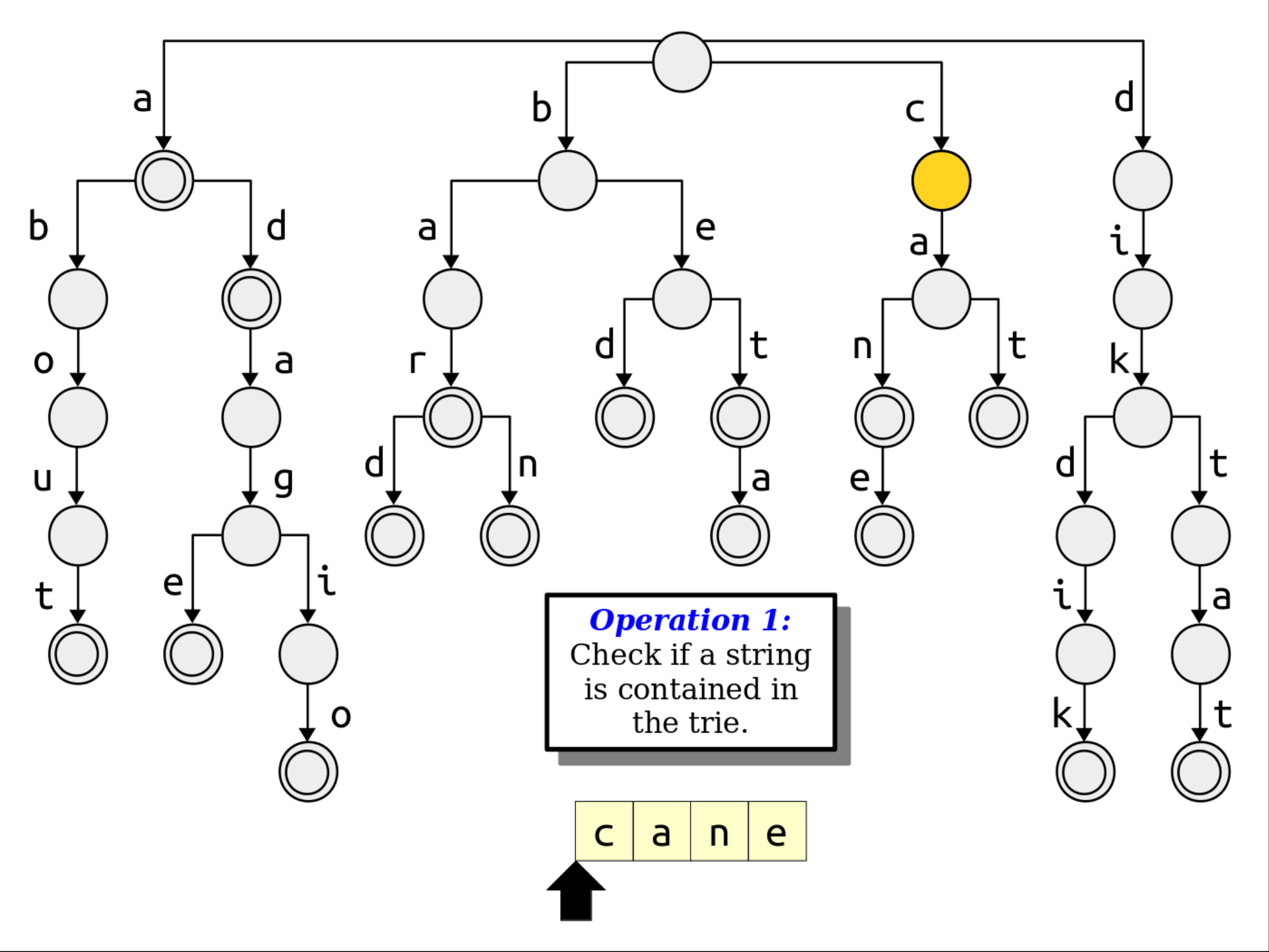
Operation 1:
Check if a string is contained in the trie.

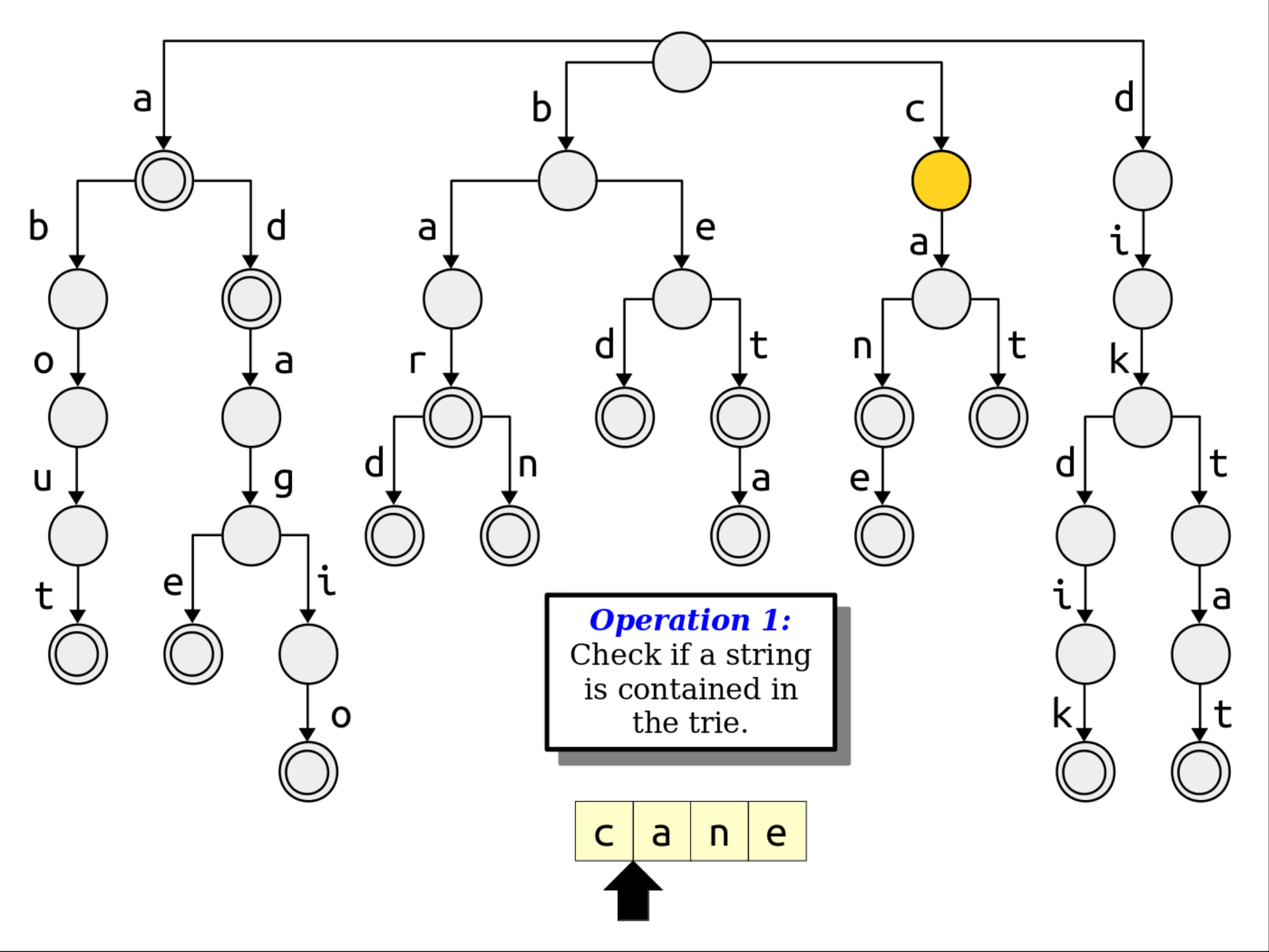


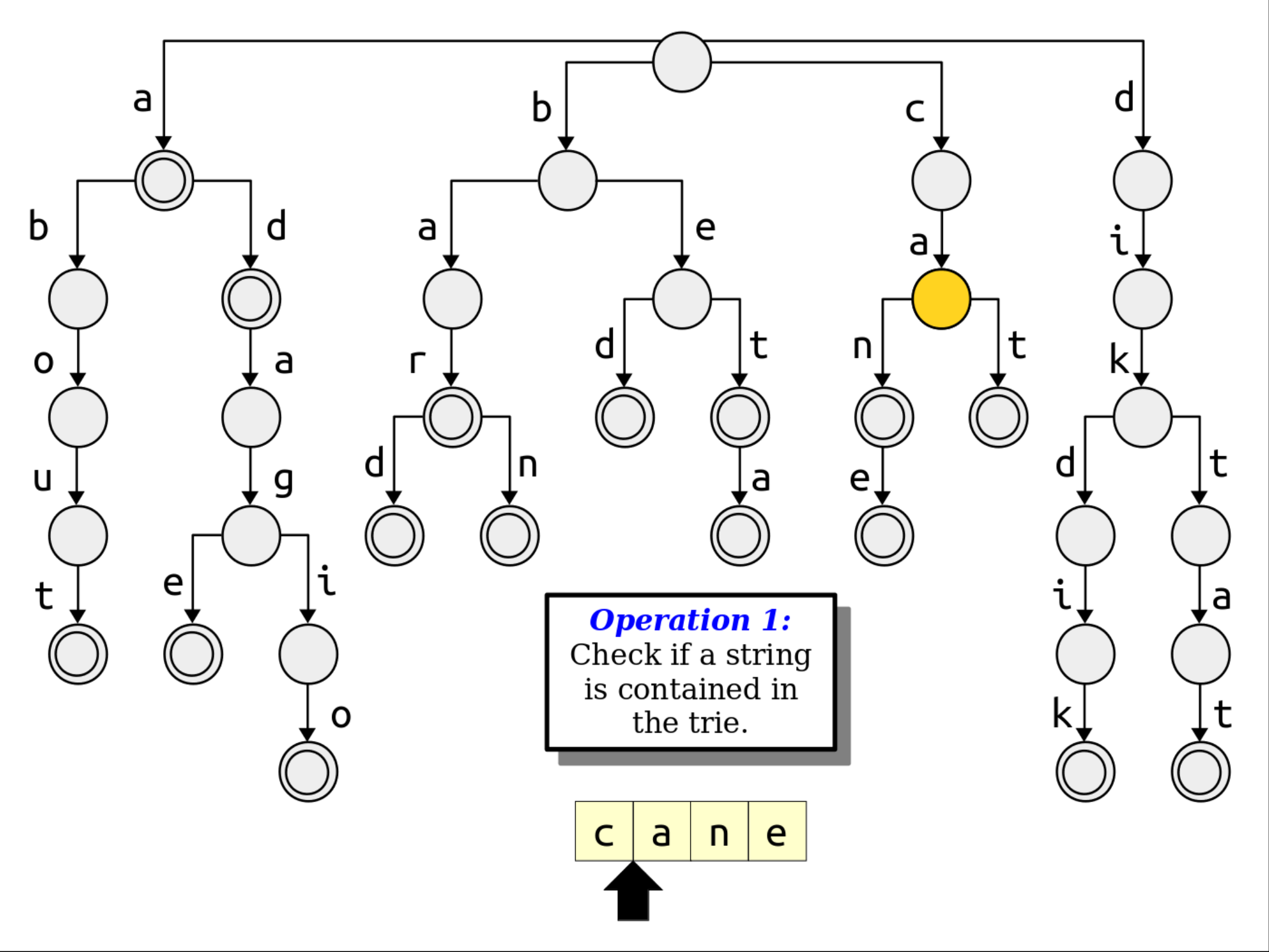
Operation 1:
Check if a string is contained in the trie.

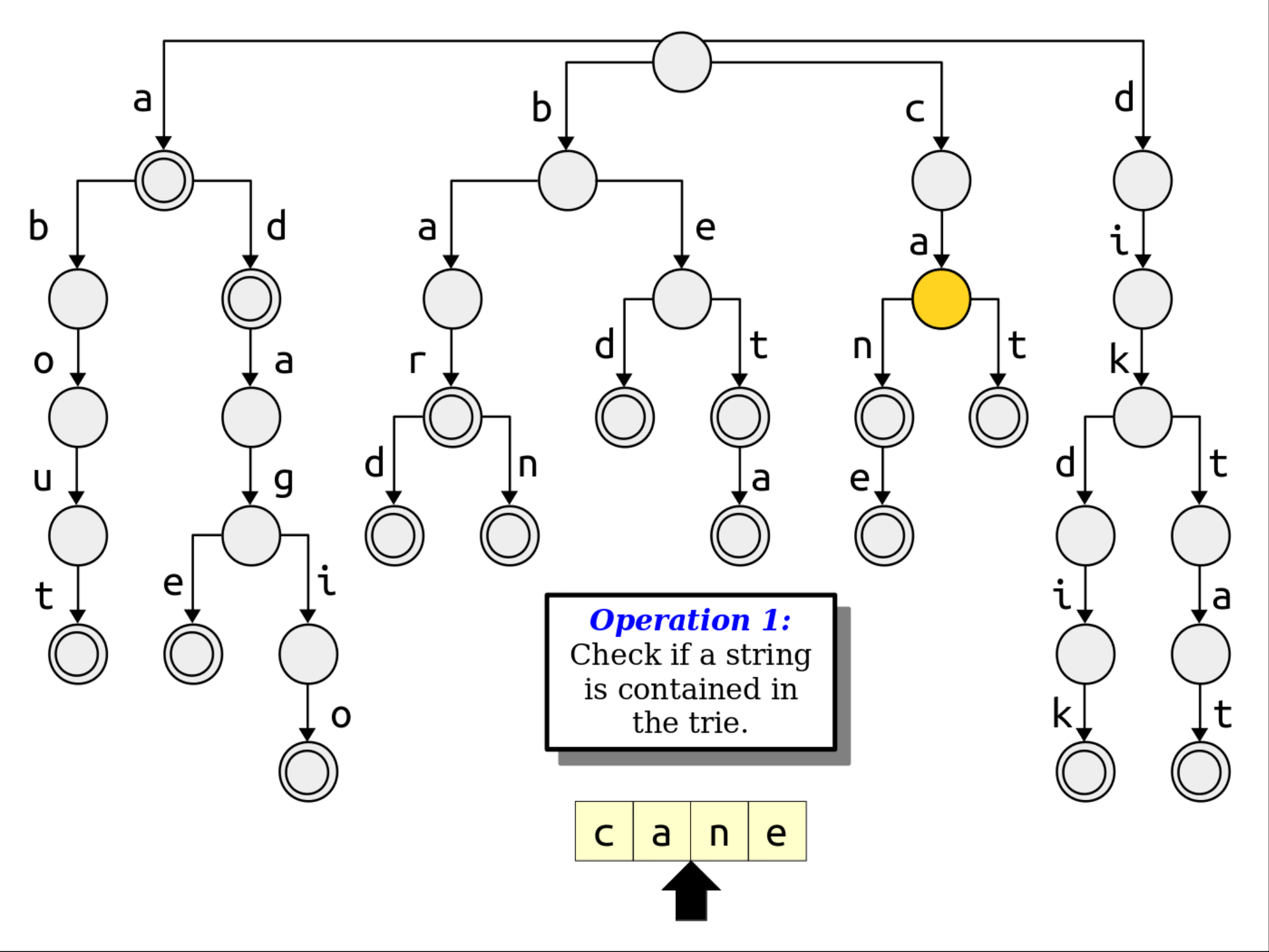
c	a	n	e


a
b
c
d
b
d
a
e
a
i
o
a
r
d
t
n
t
k
u
g
d
n
d
a
e
d
t
t
e
i
i
a
o
k
t
Operation 1:
Check if a string is contained in the trie.
c
a
n
e


a
b
c
d
b
d
a
e
a
i
o
a
r
d
t
n
t
k
u
g
d
n
d
a
e
d
t
t
e
i
i
a
o
k
t
Operation 1:
Check if a string is contained in the trie.
c
a
n
e

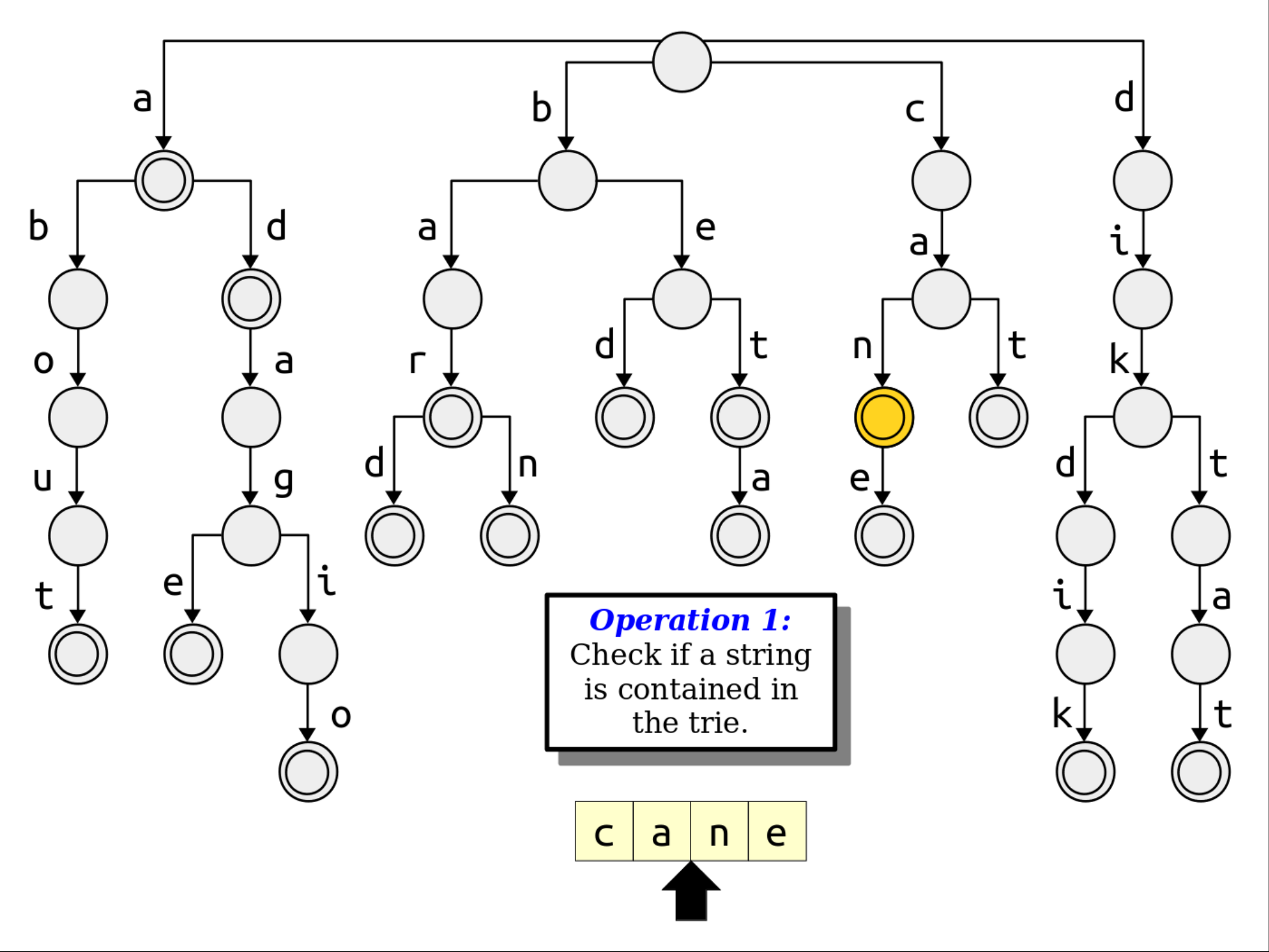

a
b
c
d
b
d
a
e
a
i
o
a
r
d
t
n
t
k
u
g
d
n
d
a
e
d
t
t
e
i
i
a
o
k
t
Operation 1:
Check if a string is contained in the trie.
c
a
n
e


a
b
c
d
b
d
a
e
a
i
o
a
r
d
t
n
t
k
u
g
d
n
d
a
e
d
t
t
e
i
i
a
o
k
t
Operation 1:
Check if a string is contained in the trie.
c
a
n
e



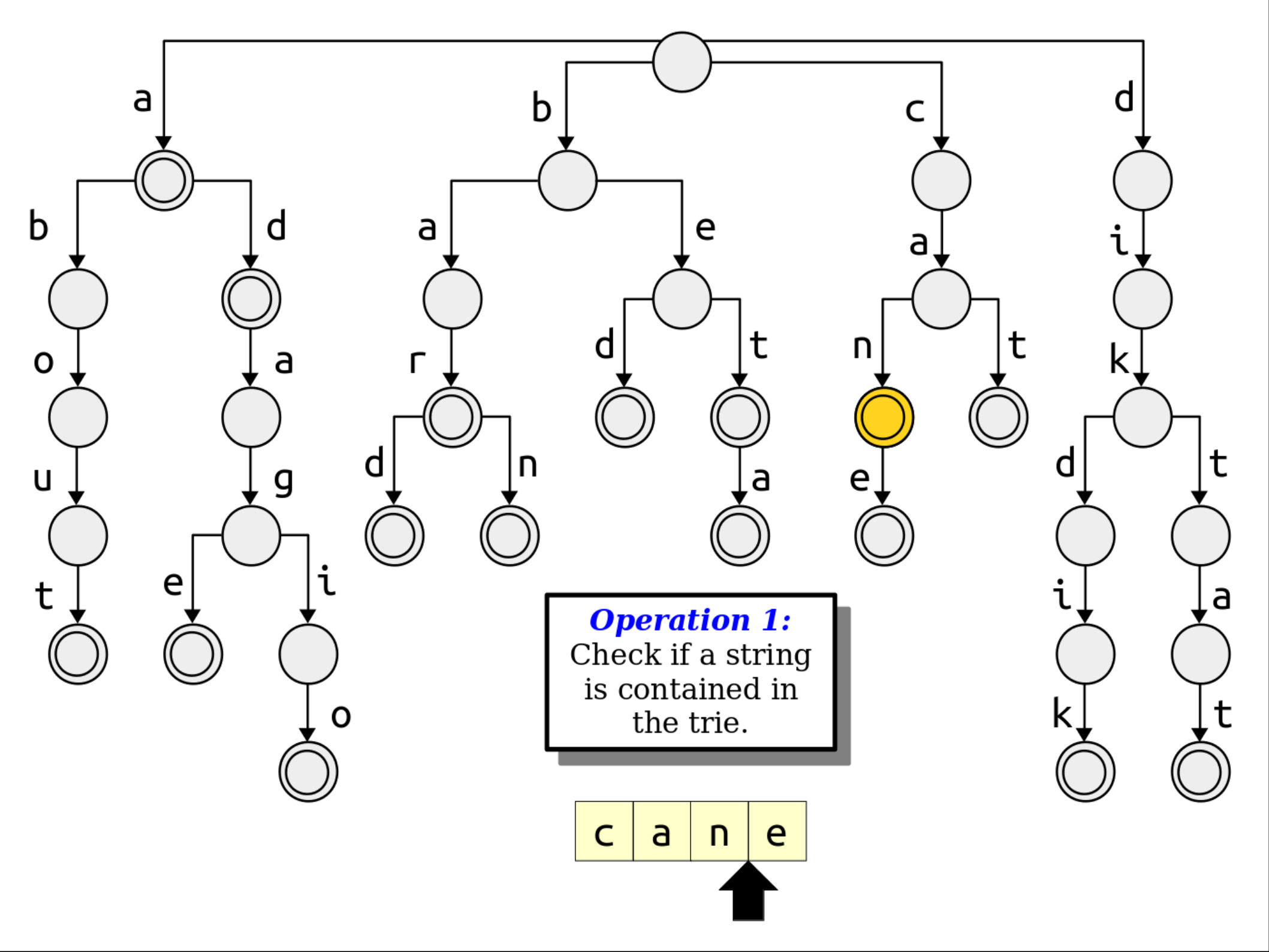
Operation 1:
Check if a string is contained in the trie.

c	a	n	e



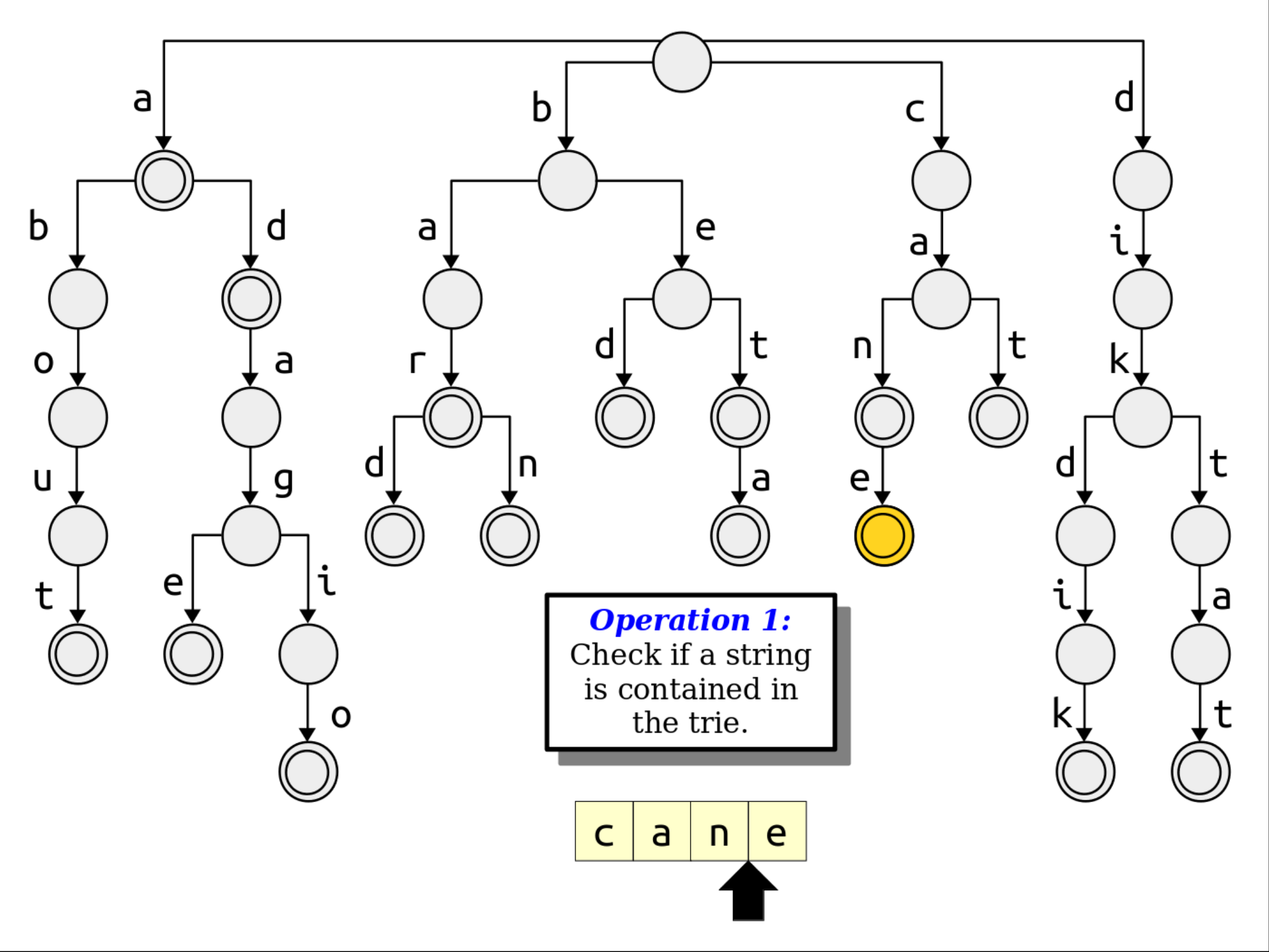
Operation 1:
Check if a string is contained in the trie.

c	a	n	e



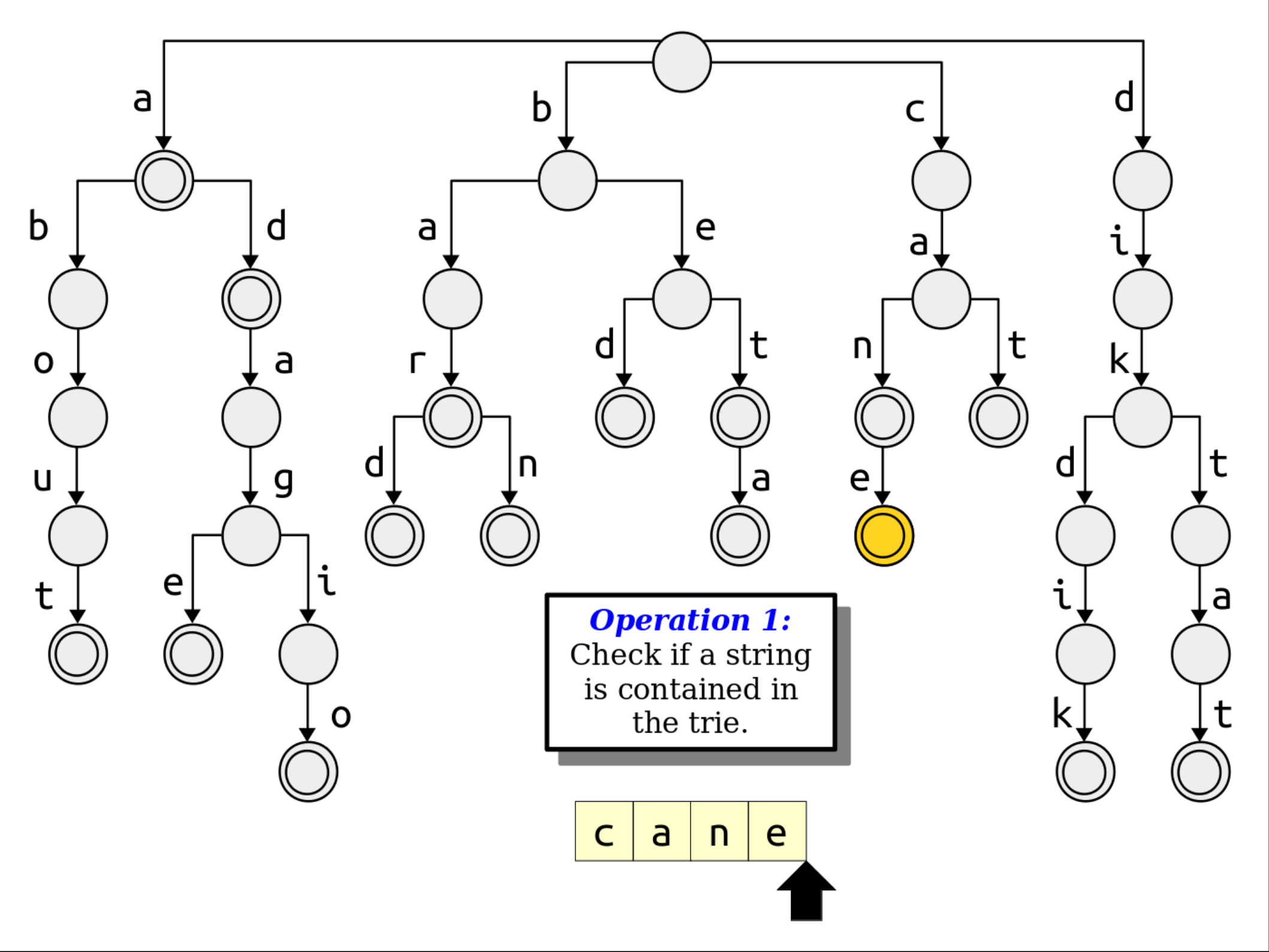
Operation 1:
Check if a string is contained in the trie.

c	a	n	e



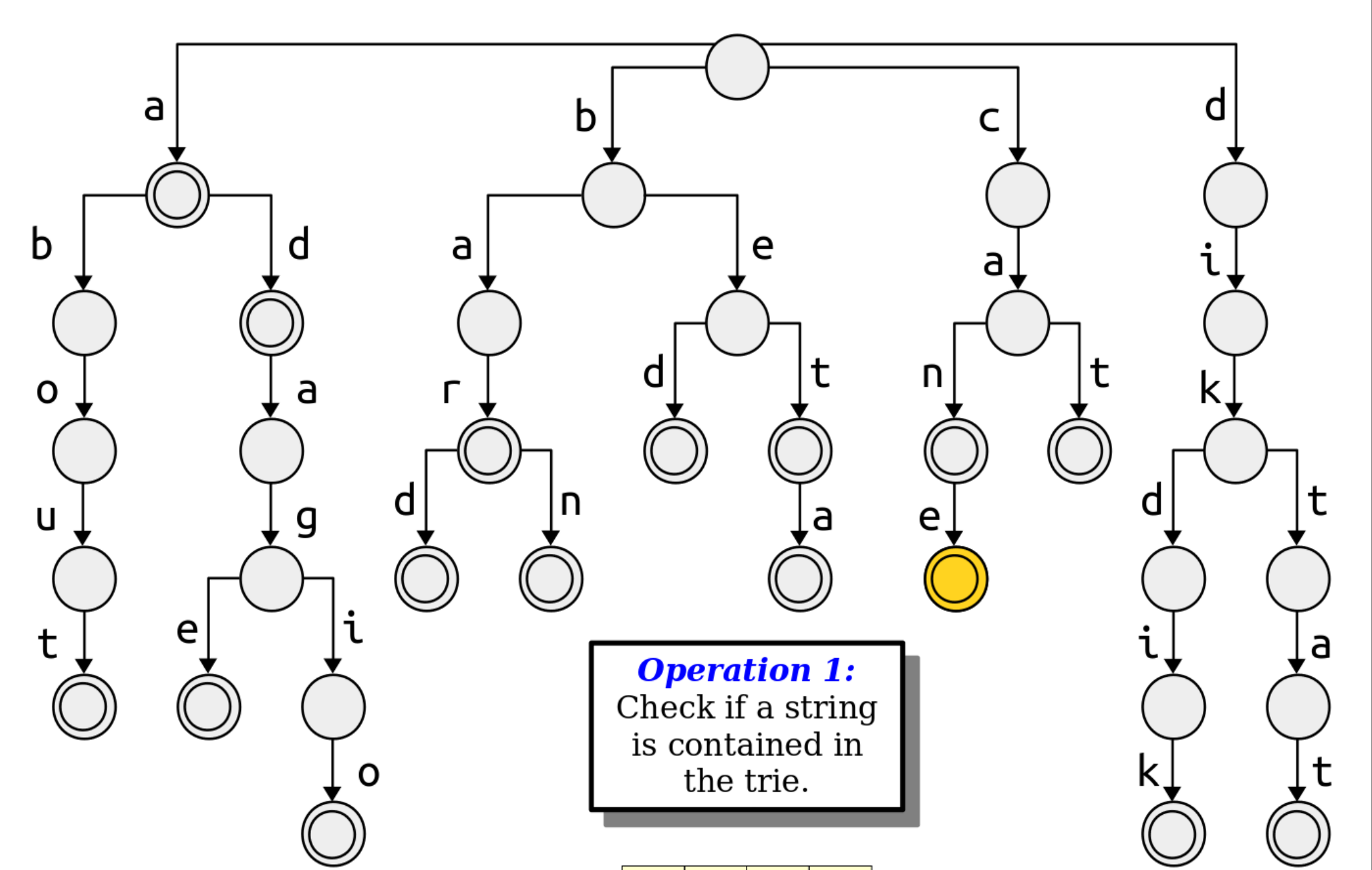
Operation 1:
Check if a string is contained in the trie.

c	a	n	e



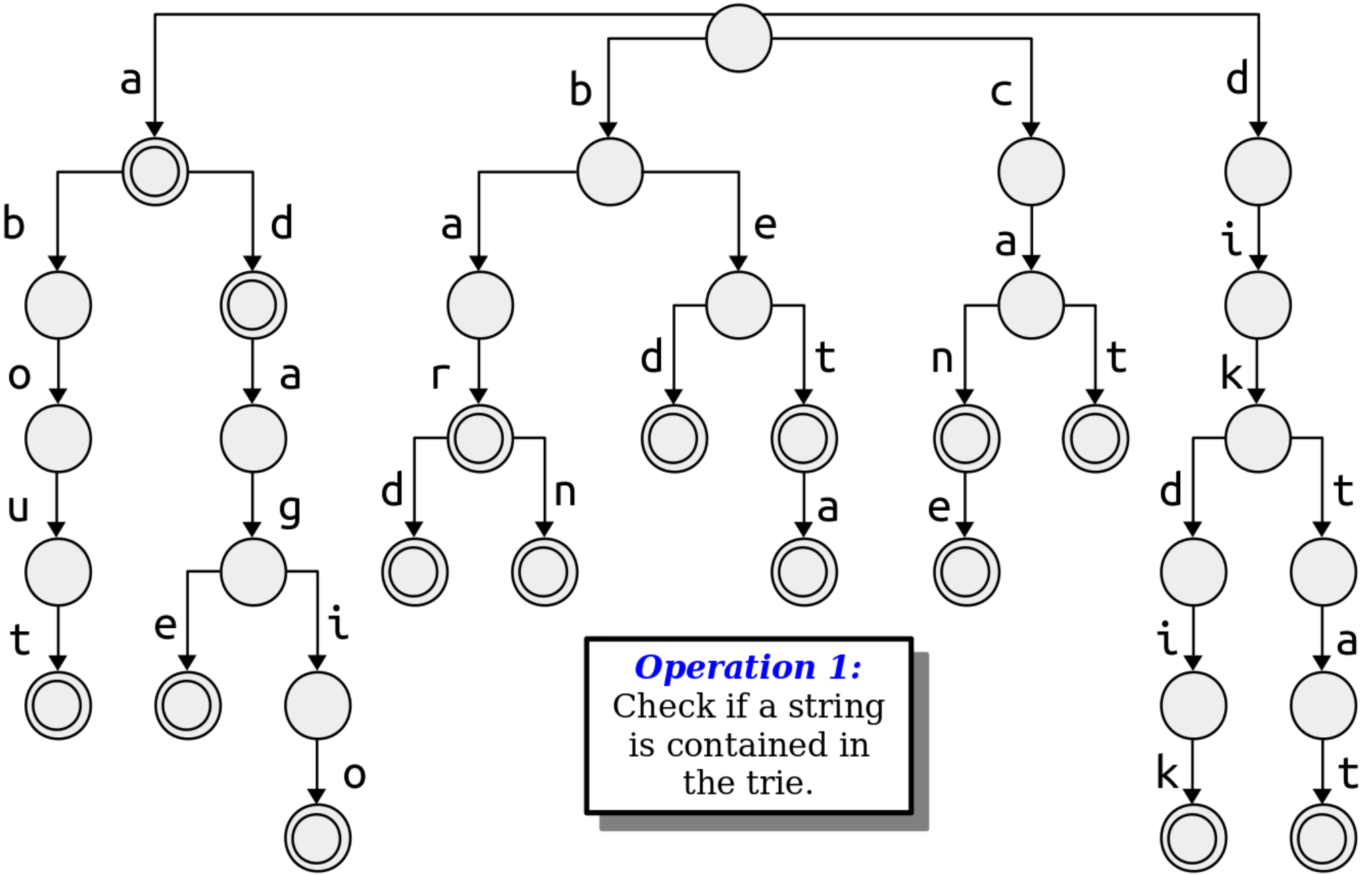
Operation 1:
Check if a string is contained in the trie.

c	a	n	e

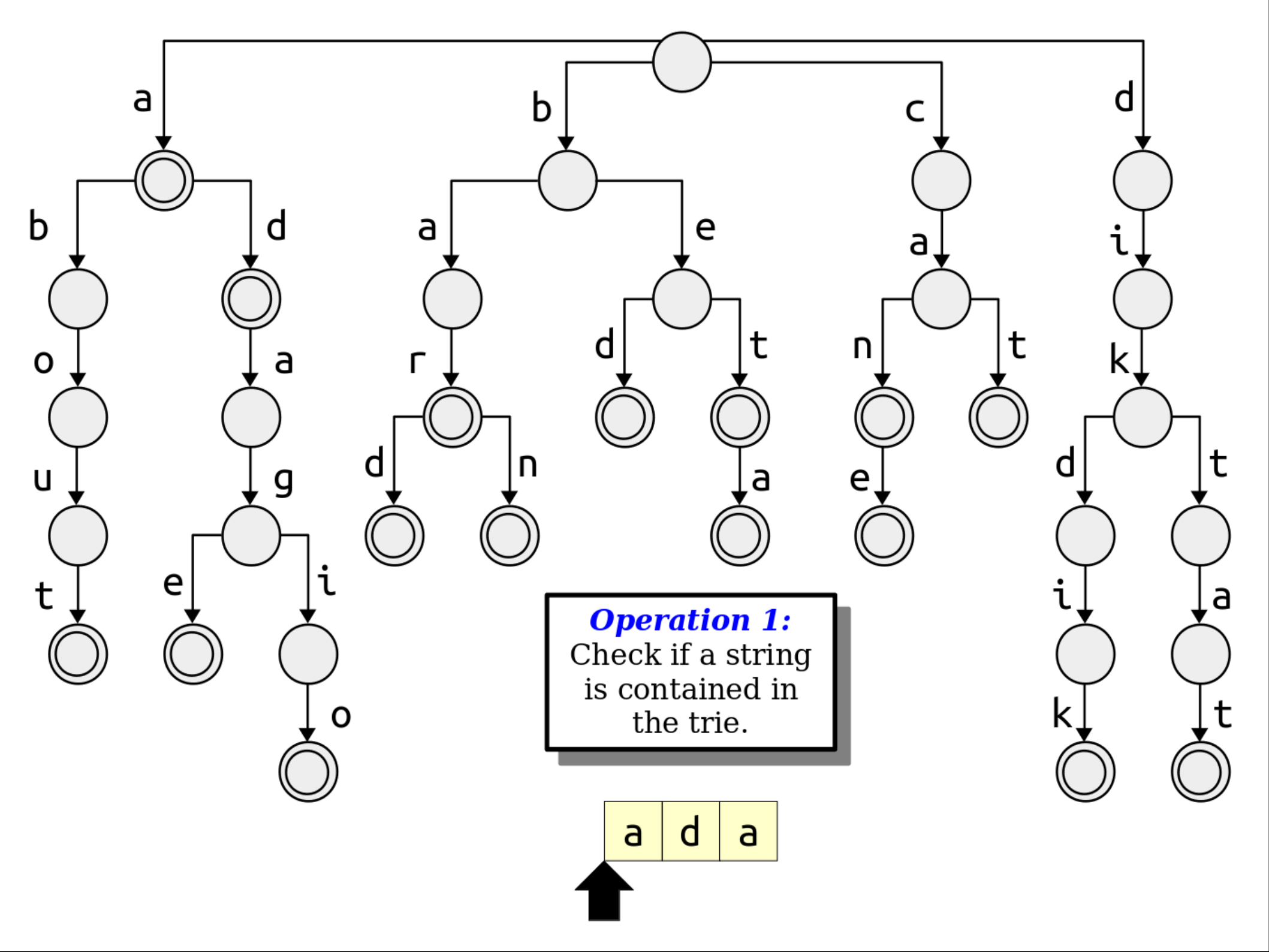


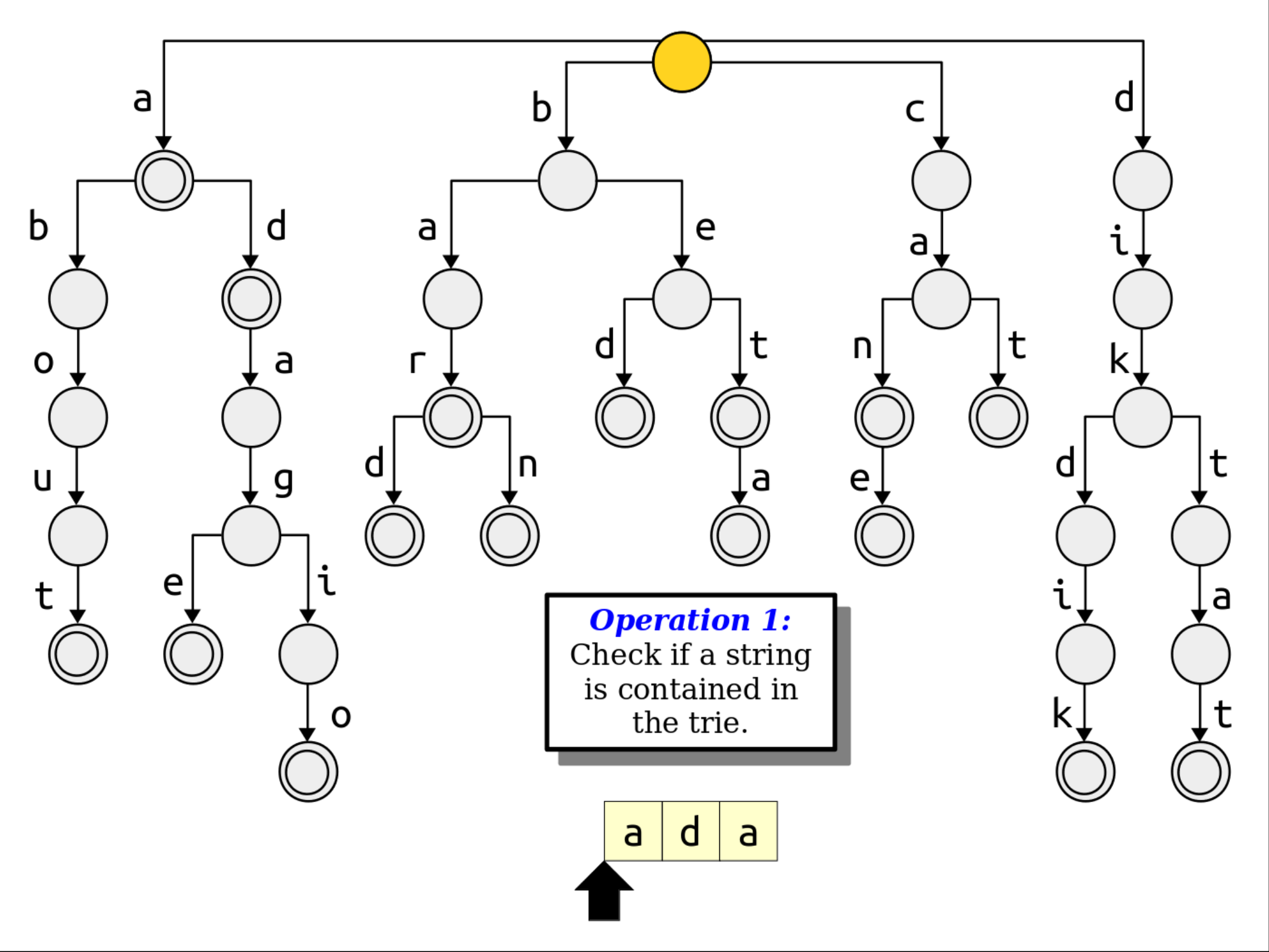
Operation 1:
Check if a string is contained in the trie.

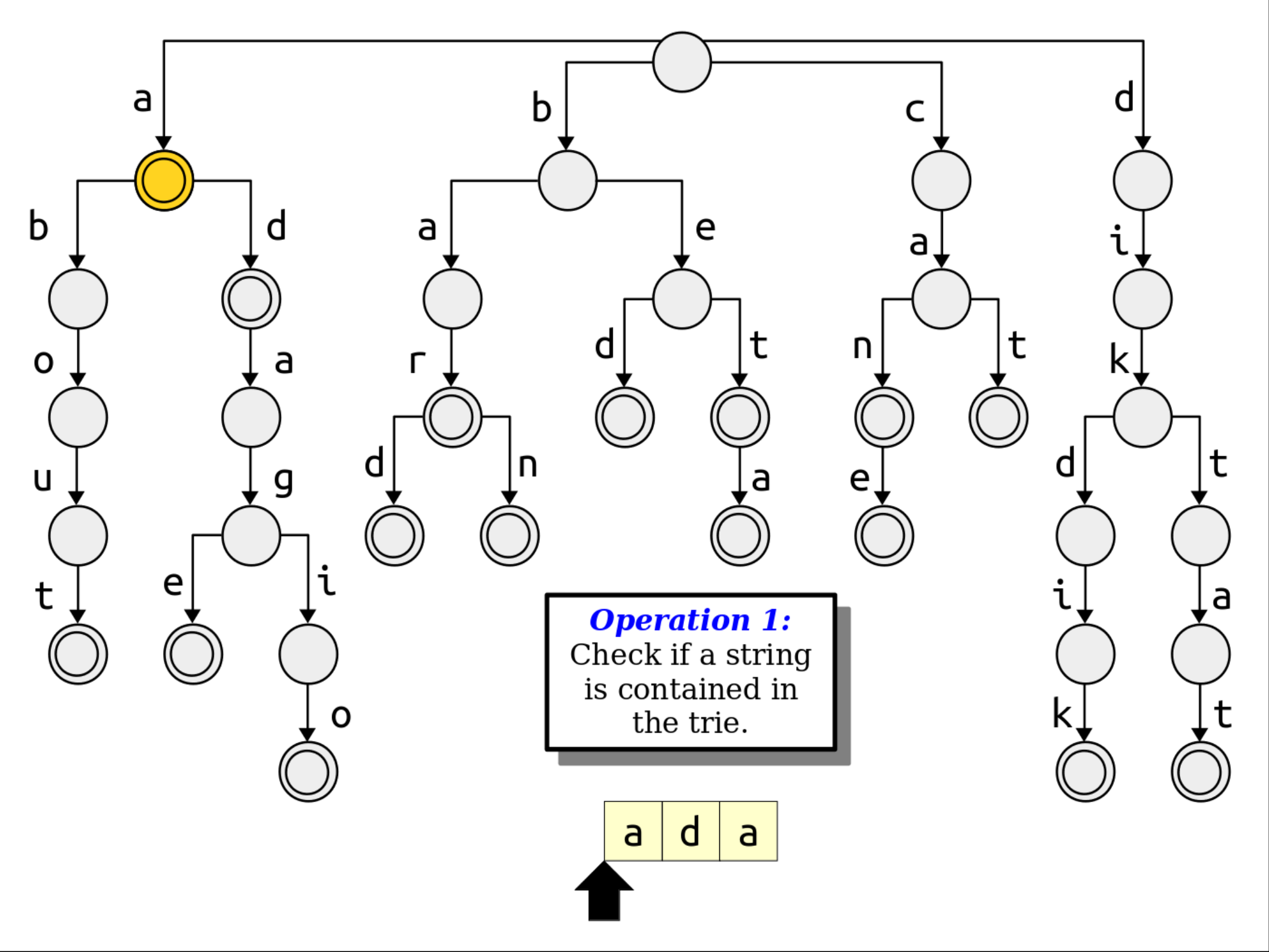
c	a	n	e

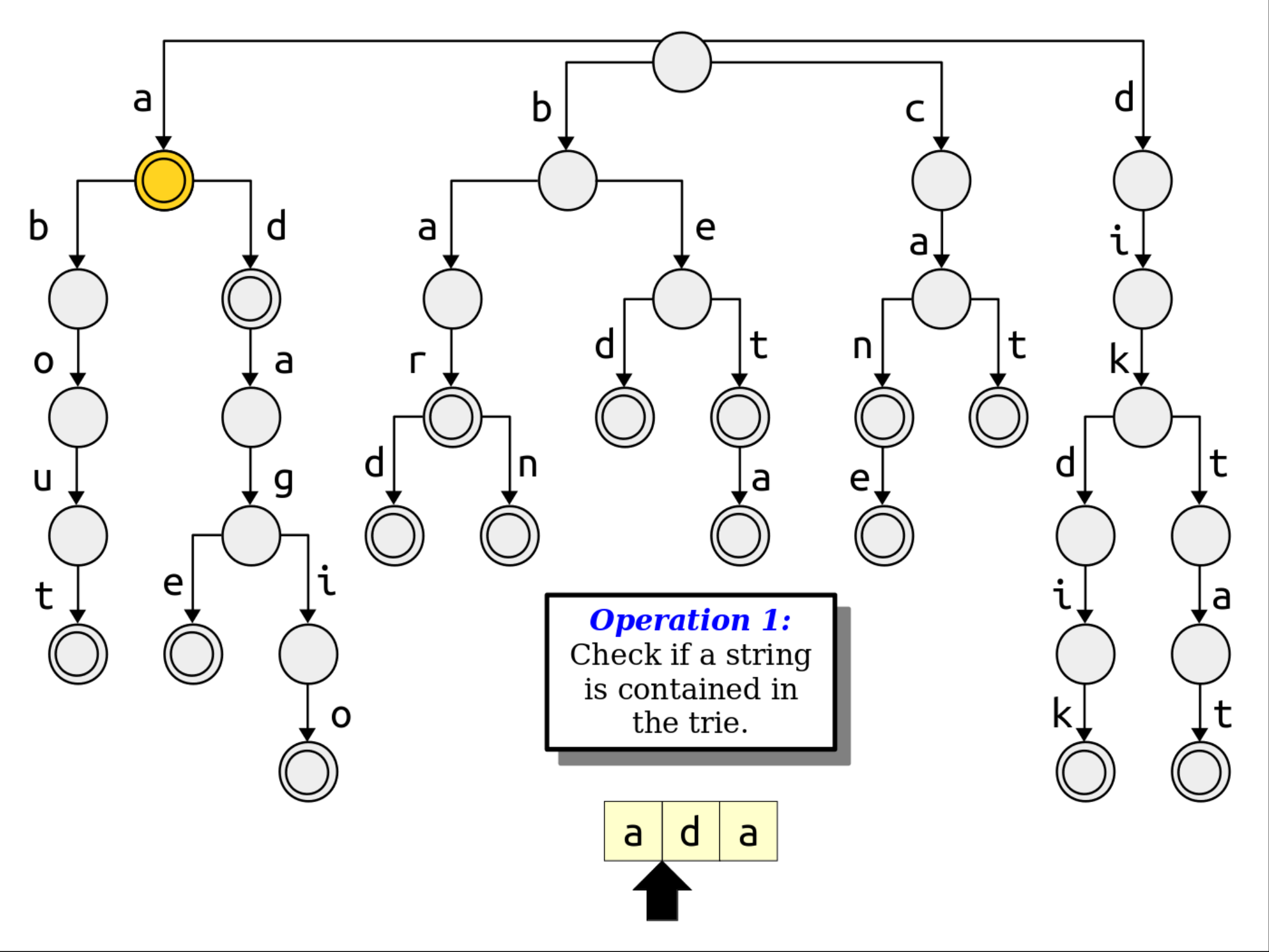


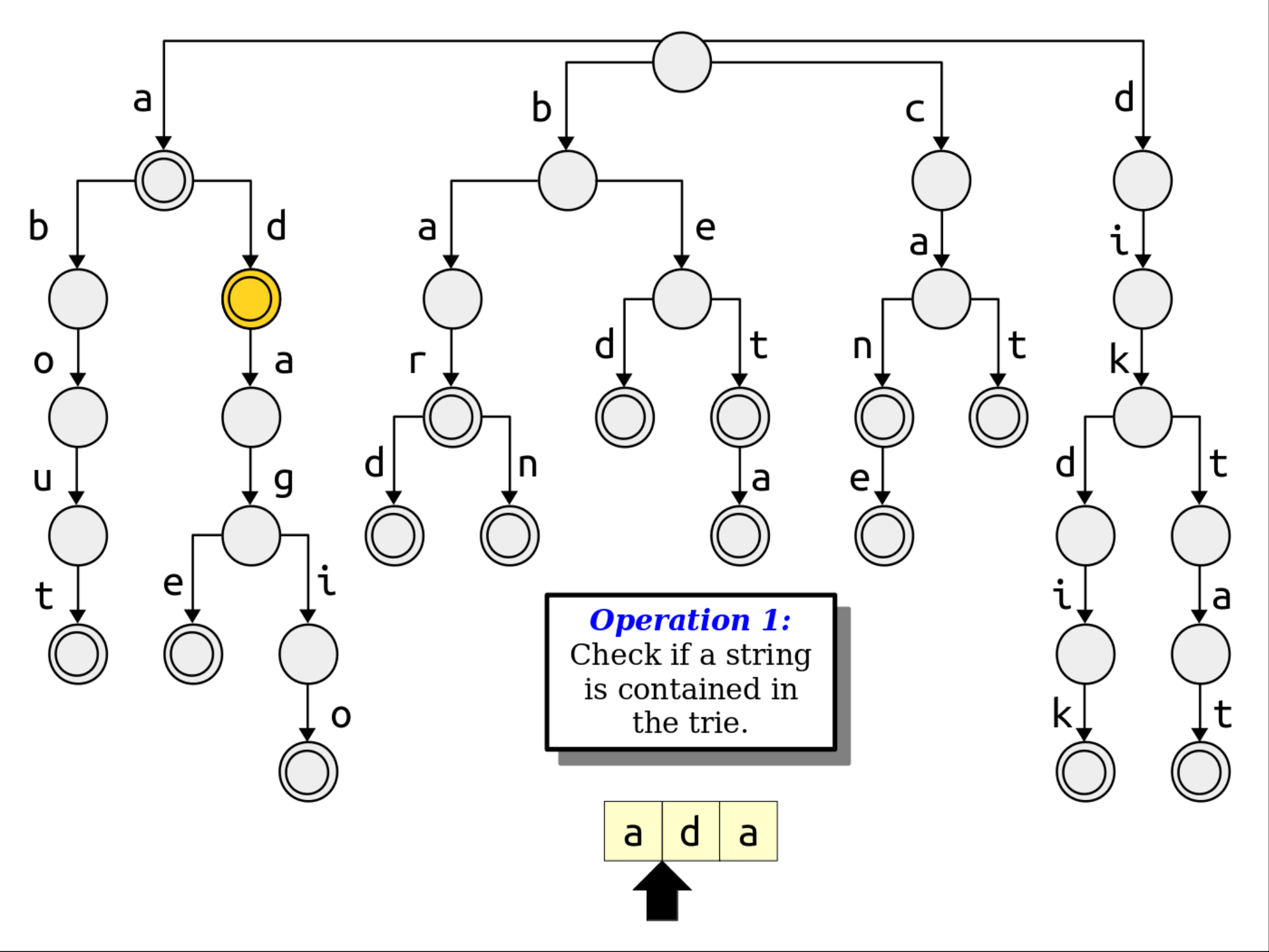
Operation 1:
Check if a string is contained in the trie.


a
b
c
d
b
d
a
e
a
i
o
a
r
d
t
n
t
k
u
g
d
n
d
t
a
e
t
e
i
d
t
i
a
o
i
k
t
Operation 1:
Check if a string is contained in the trie.
a
d
a


a
b
c
d
b
d
a
e
a
i
o
a
r
d
t
n
t
k
u
g
d
n
d
t
a
e
t
e
i
d
t
i
a
o
i
k
t
Operation 1:
Check if a string is contained in the trie.
a
d
a

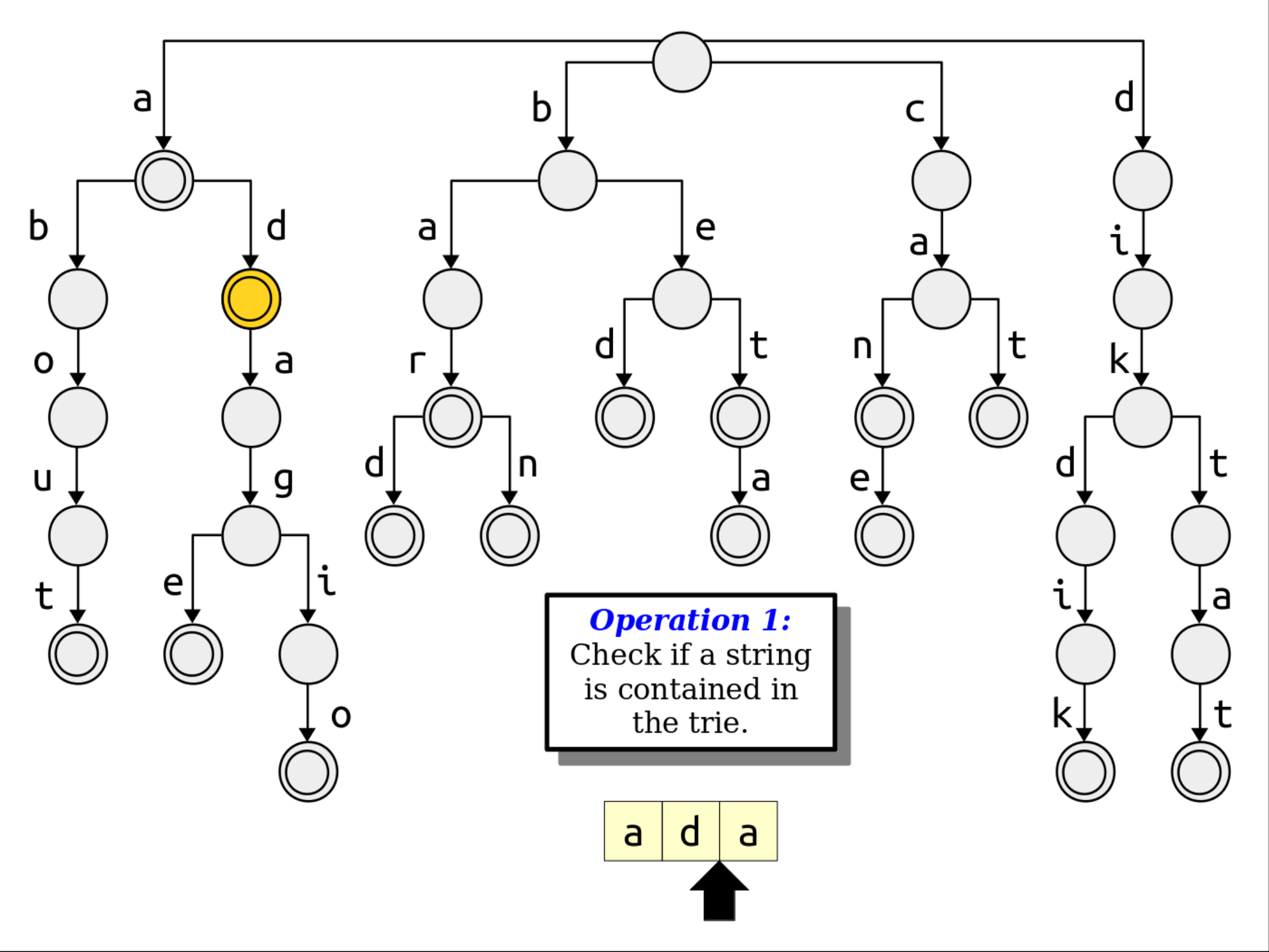

a
b
c
d
b
d
a
e
a
i
o
a
r
d
t
n
t
k
u
g
d
n
d
t
a
e
t
e
i
d
t
i
a
o
i
k
t
Operation 1:
Check if a string is contained in the trie.
a
d
a


a
b
c
d
b
d
a
e
a
i
o
a
r
d
t
n
t
k
u
g
d
n
d
t
a
e
d
t
t
e
i
i
a
o
k
t
k
t
Operation 1:
Check if a string is contained in the trie.
a
d
a

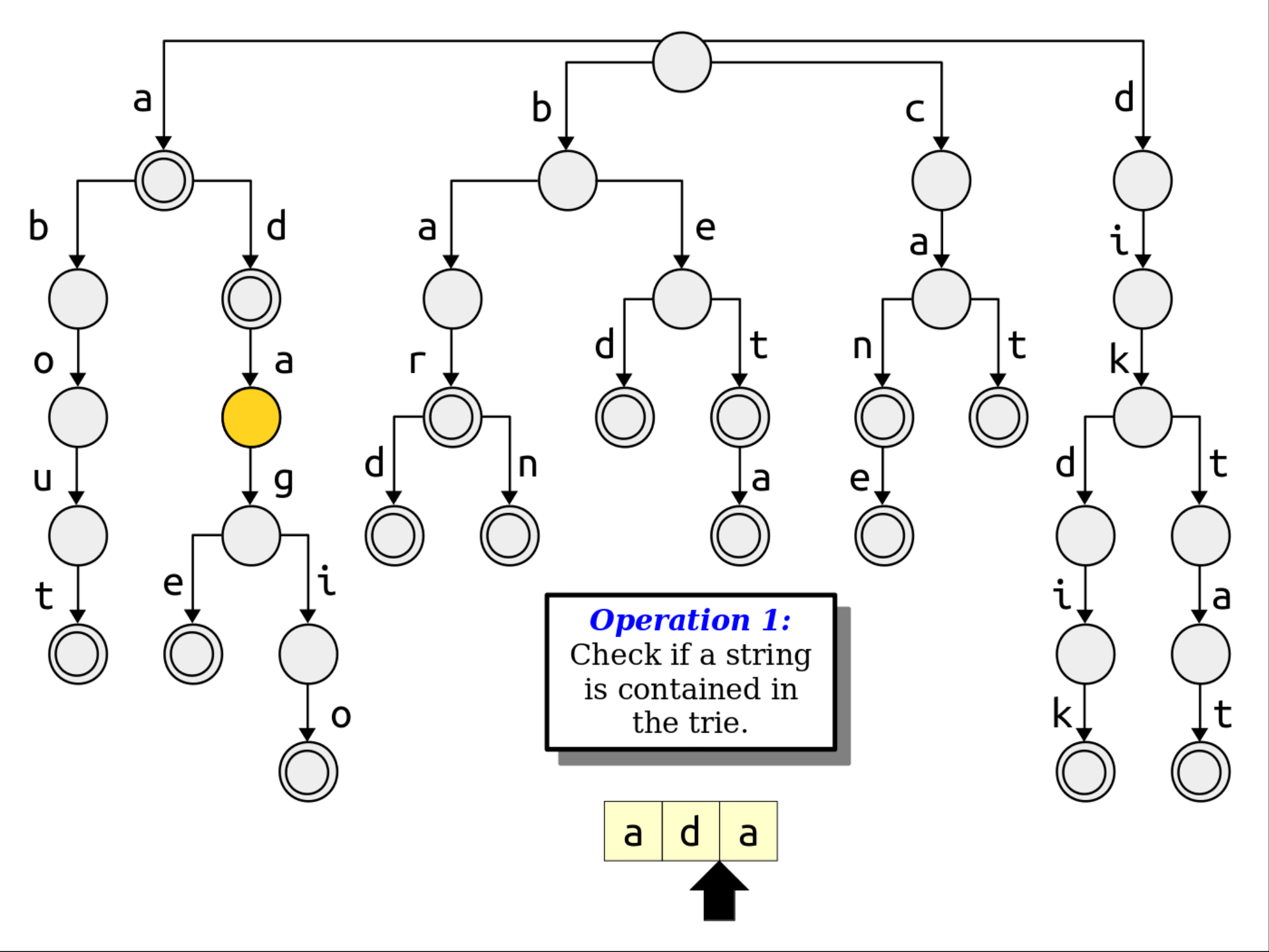


Operation 1:
Check if a string is contained in the trie.

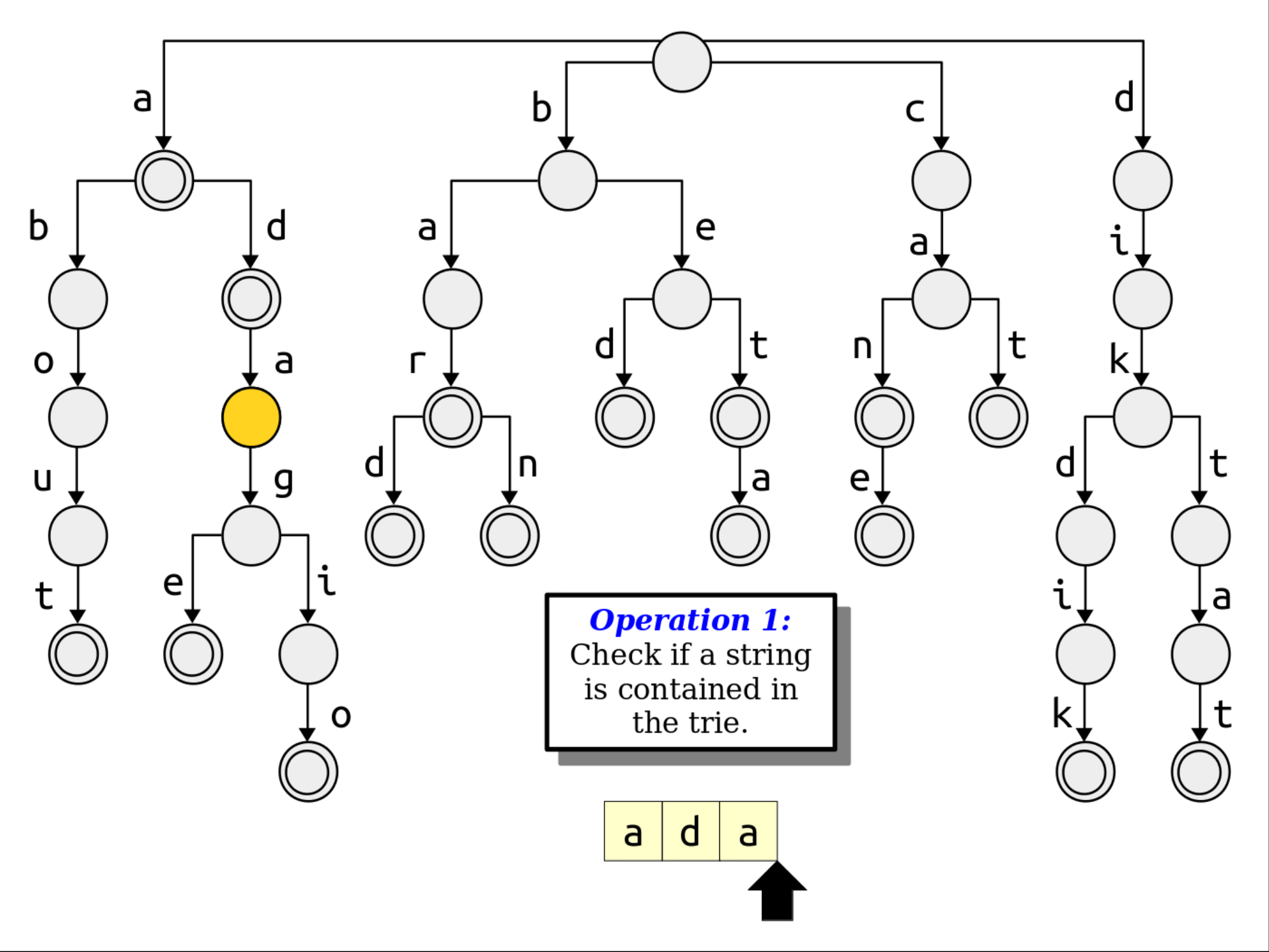
a	d	a

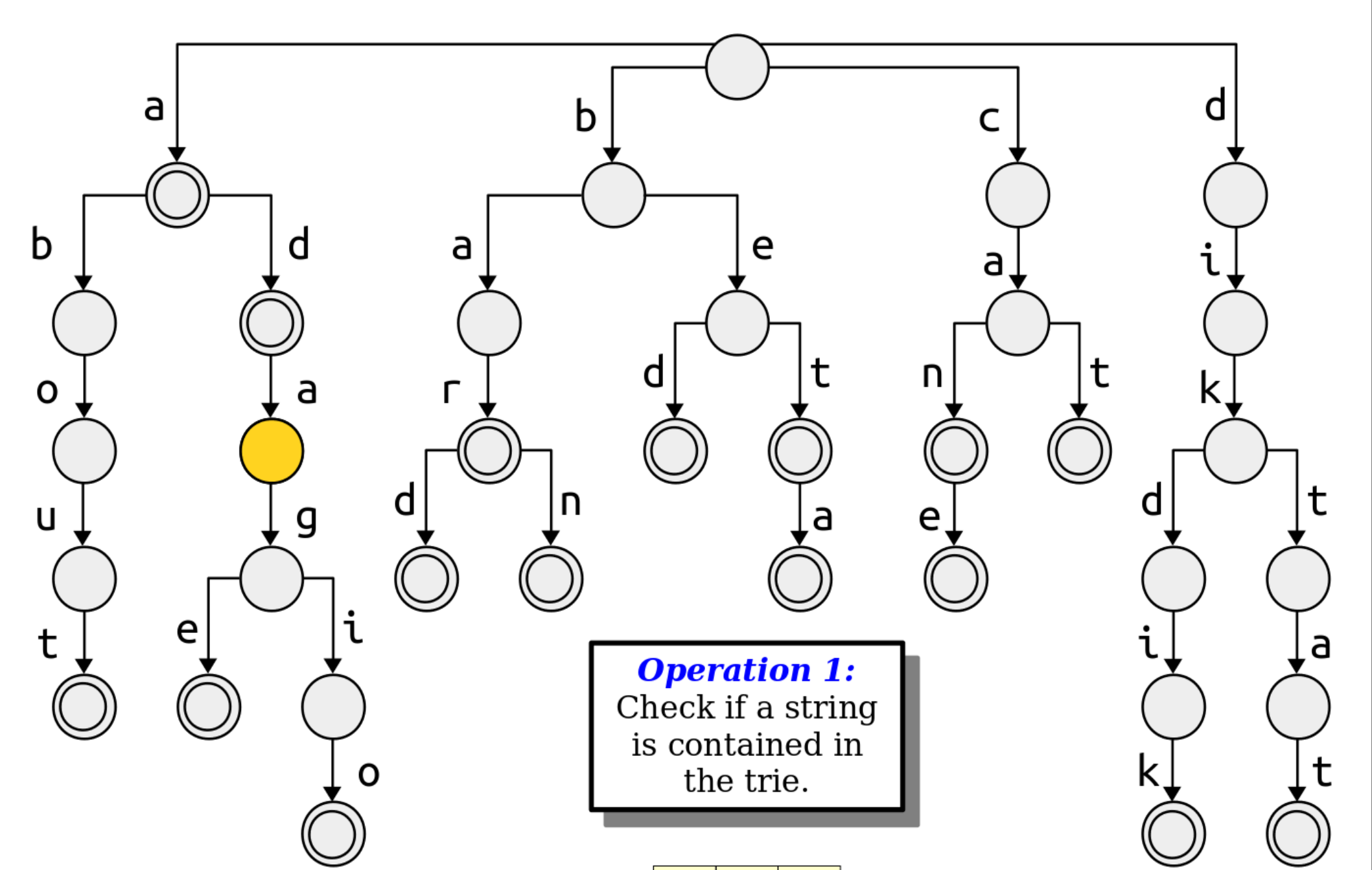


Operation 1:
Check if a string is contained in the trie.



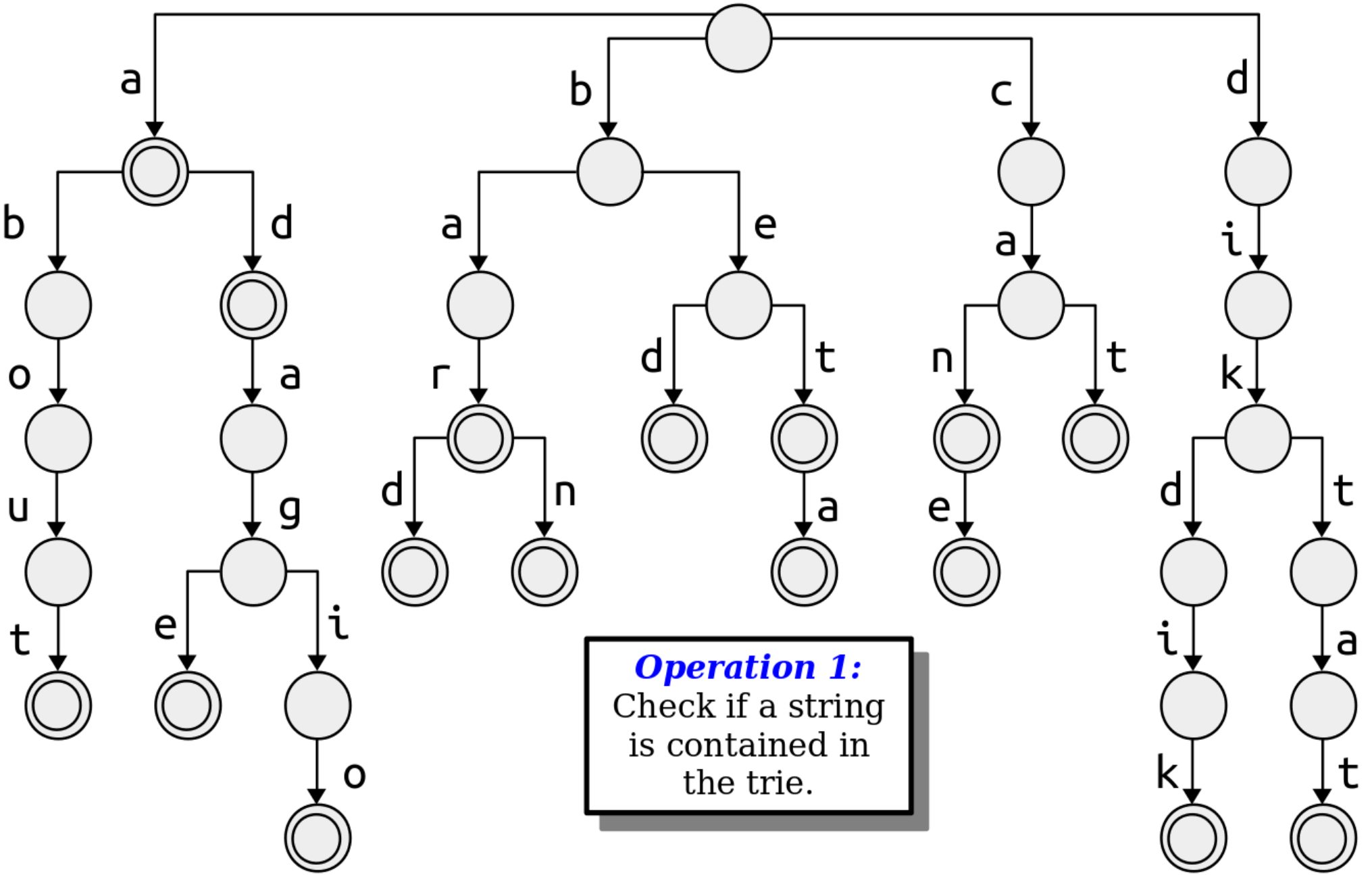
Operation 1:
Check if a string is contained in the trie.


a
b
c
d
b
d
a
e
a
i
o
a
r
d
t
n
t
k
u
g
d
n
d
t
a
e
d
t
t
e
i
i
a
o
k
t
k
t
Operation 1:
Check if a string is contained in the trie.
a d a

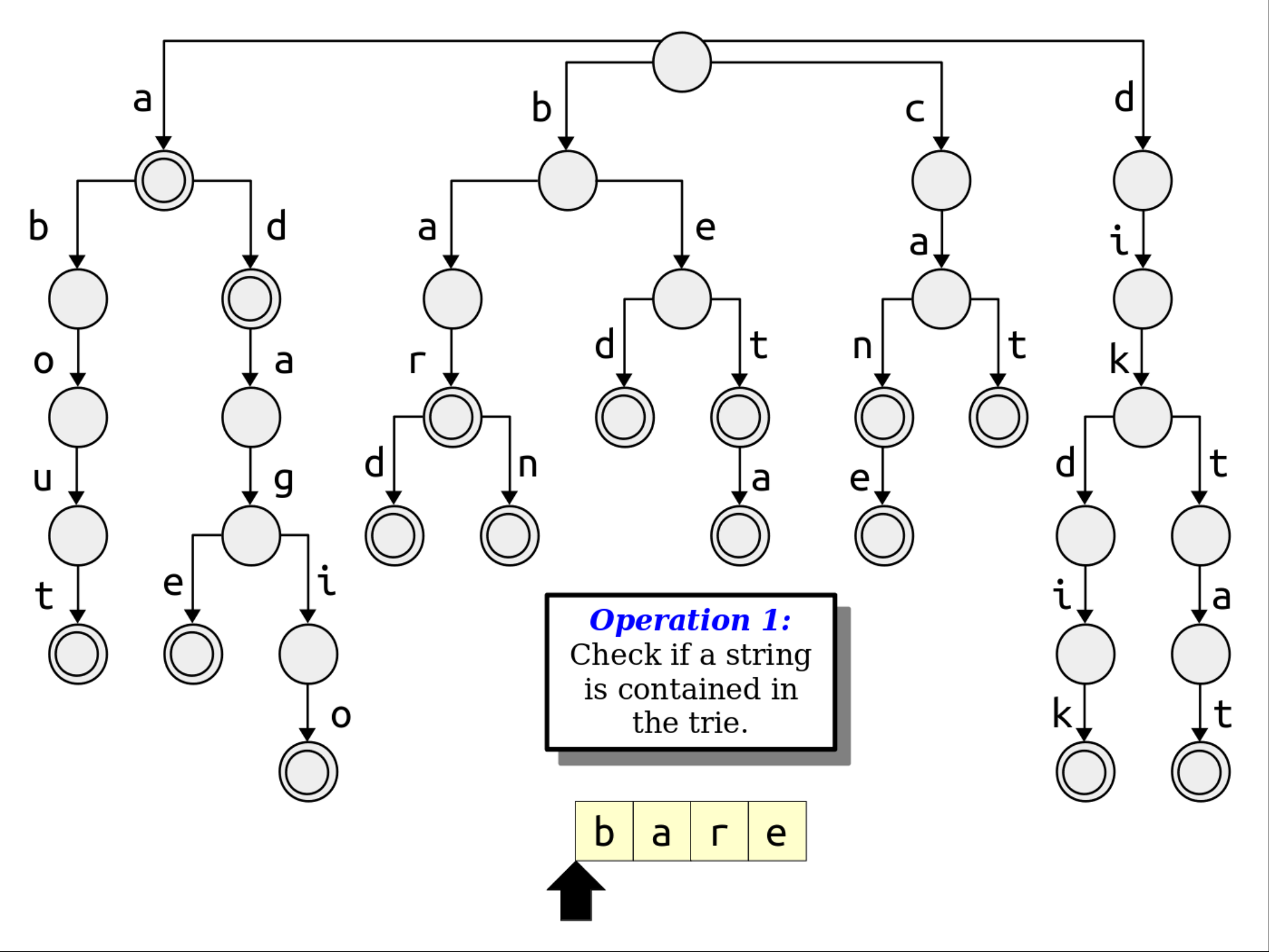


Operation 1:
Check if a string is contained in the trie.

a	d	a

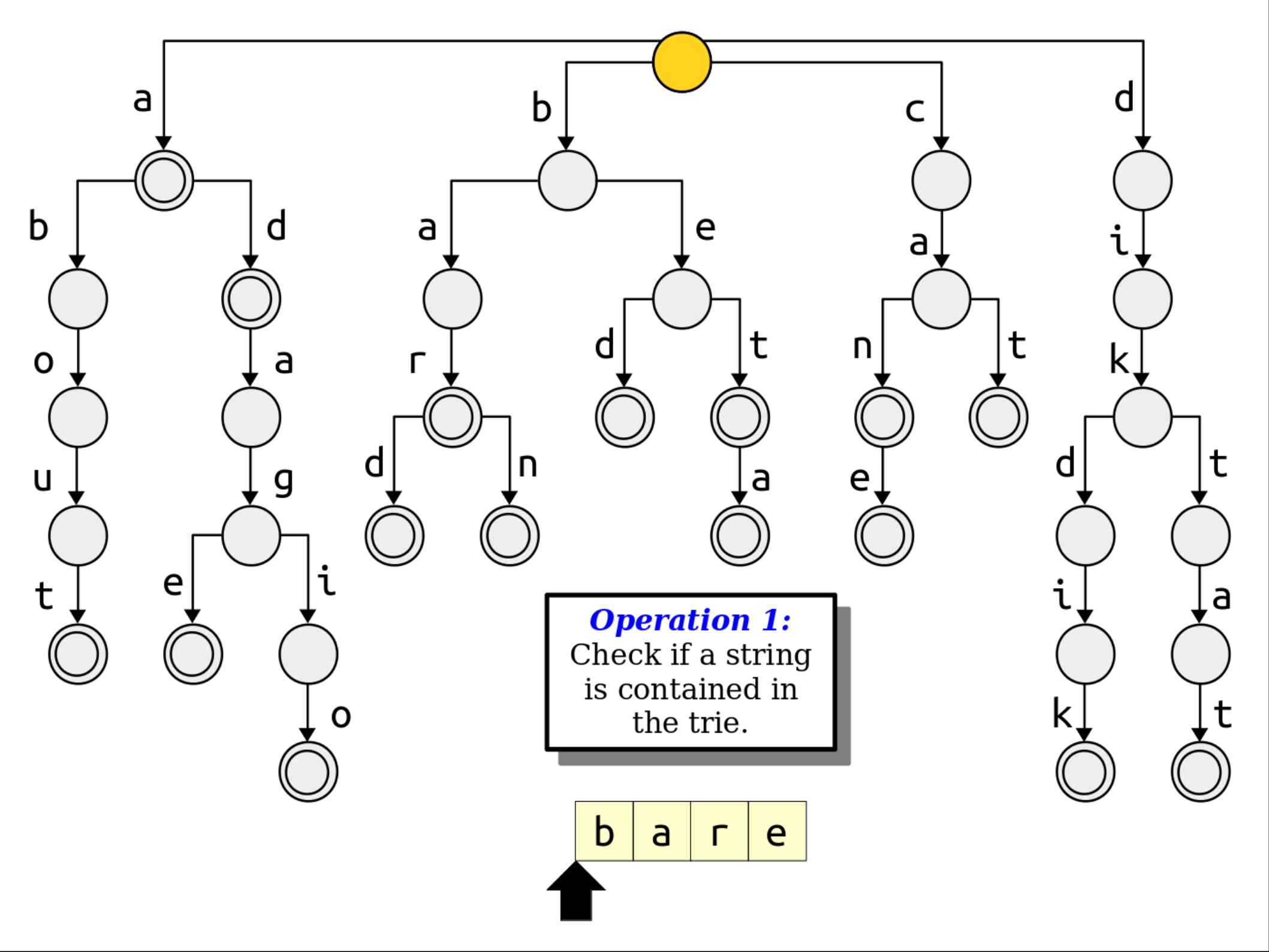


Operation 1:
Check if a string is contained in the trie.



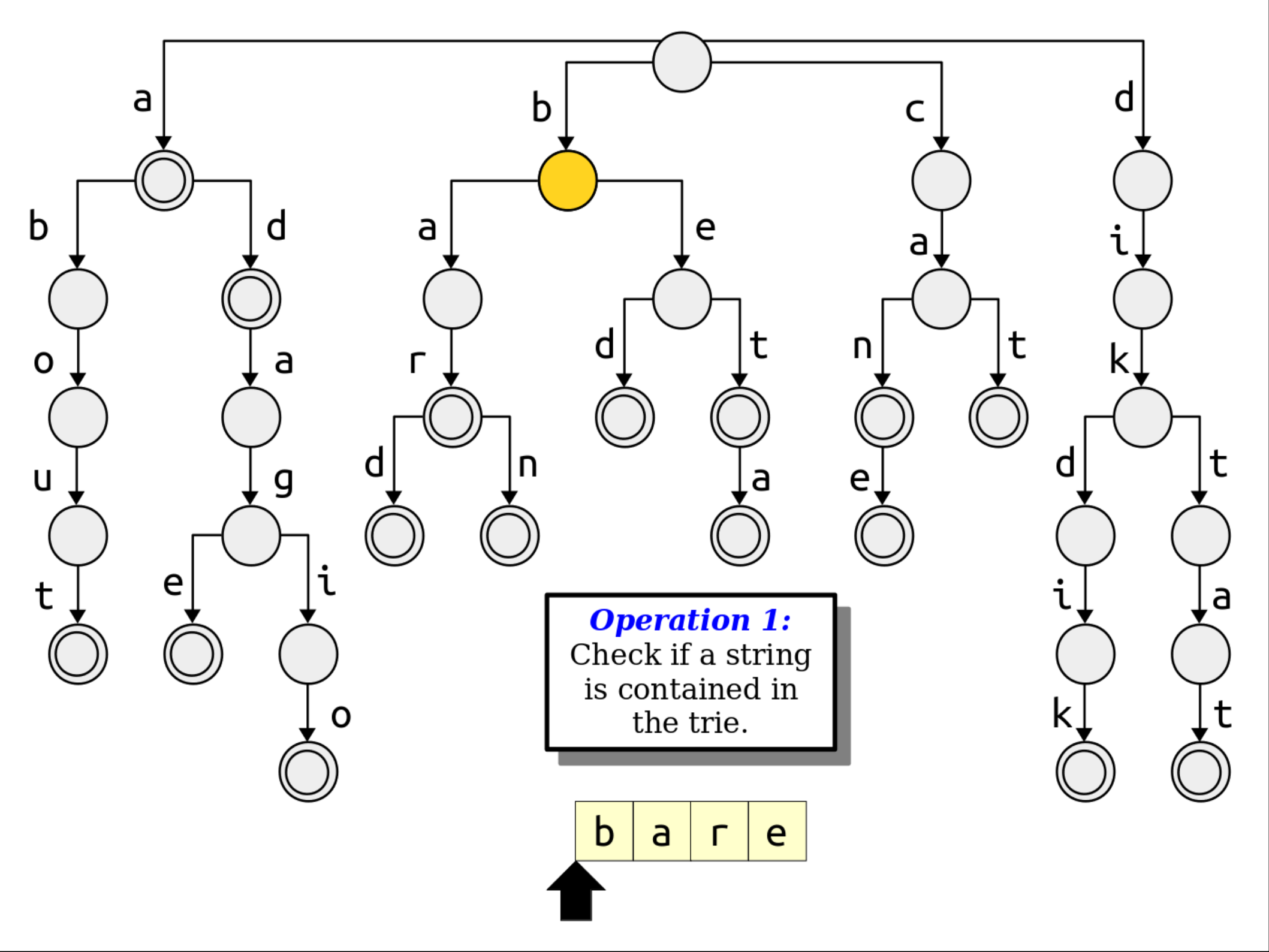
Operation 1:
Check if a string is contained in the trie.

b	a	r	e



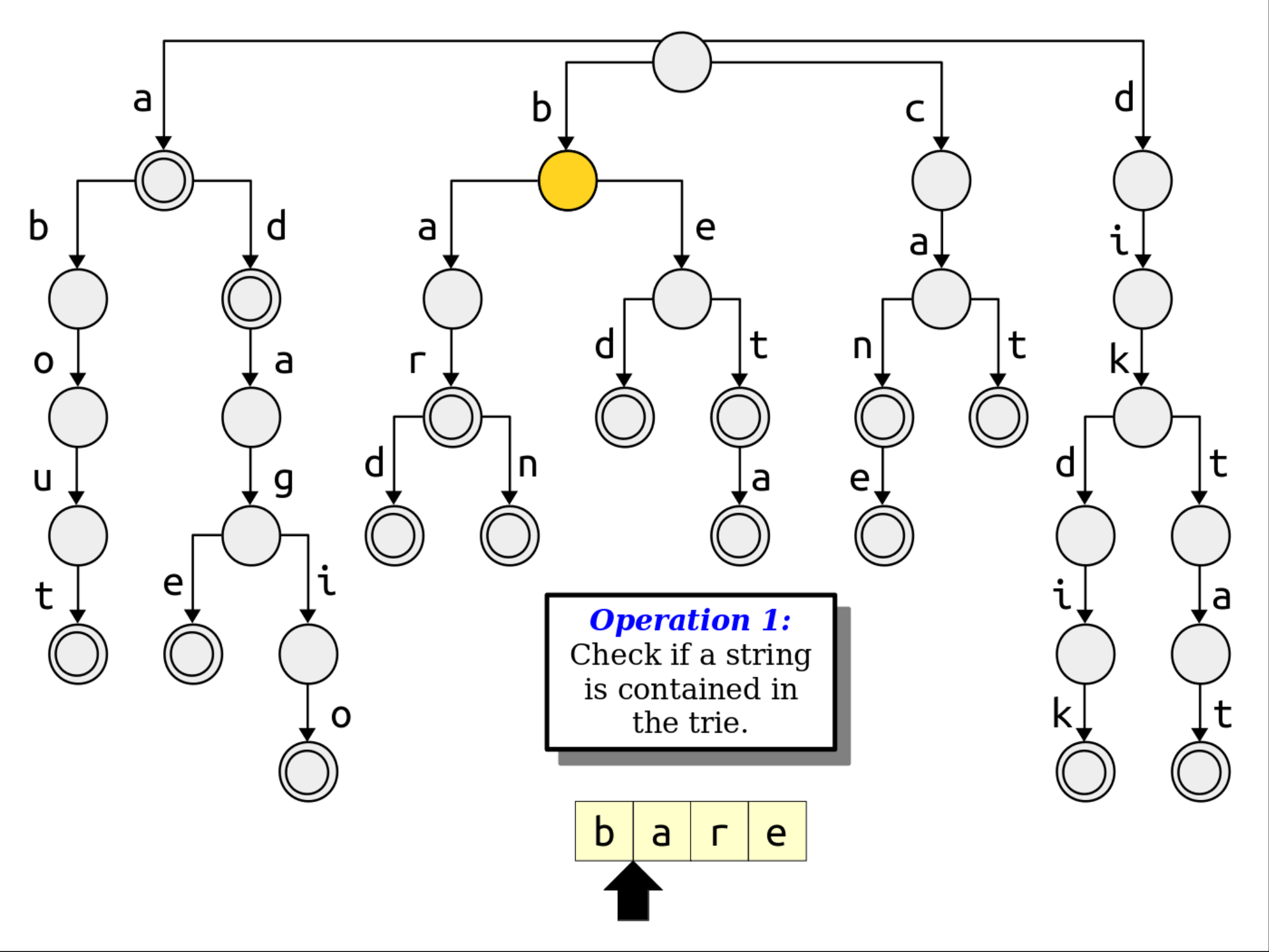
Operation 1:
Check if a string is contained in the trie.

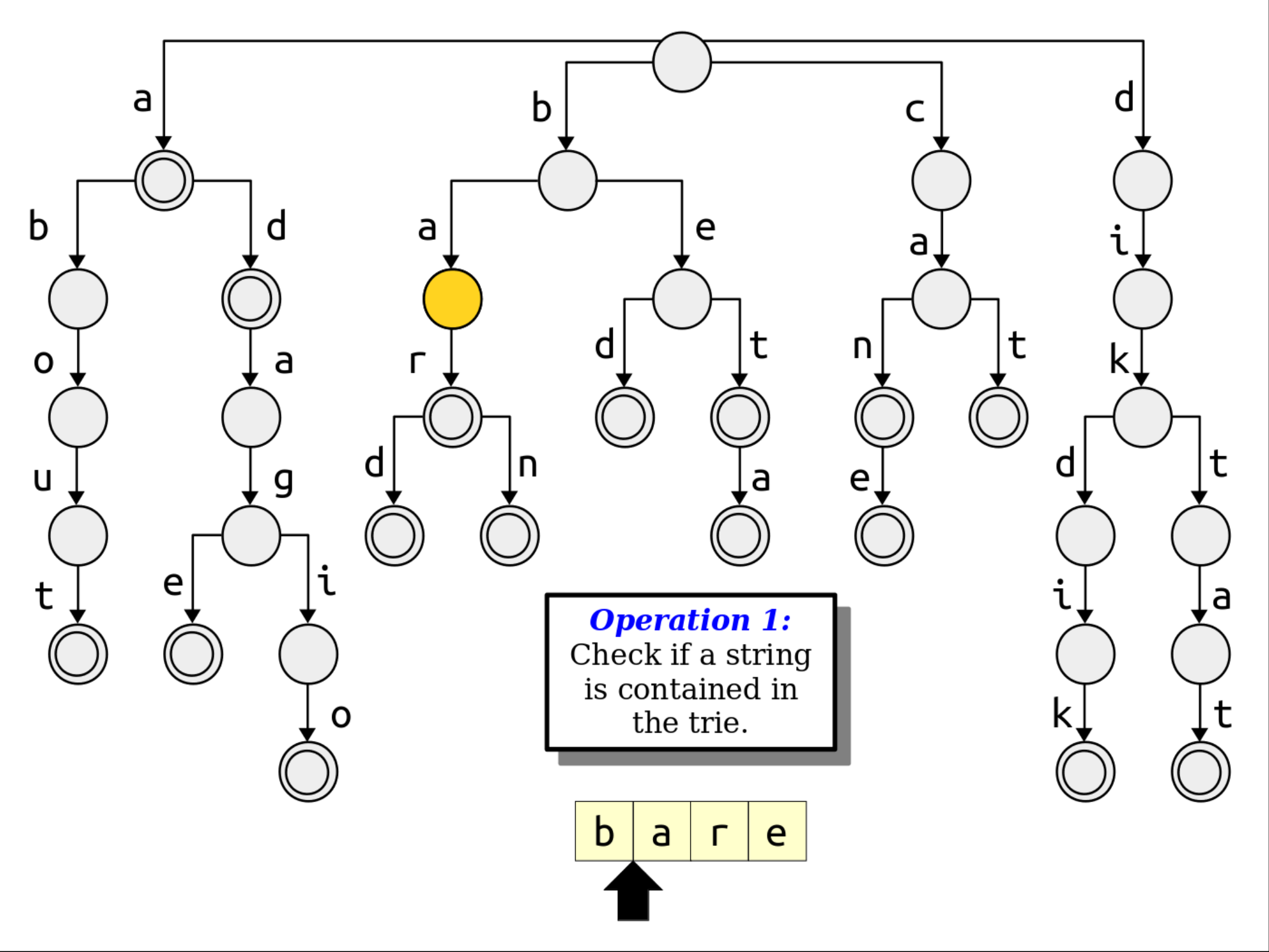
b	a	r	e

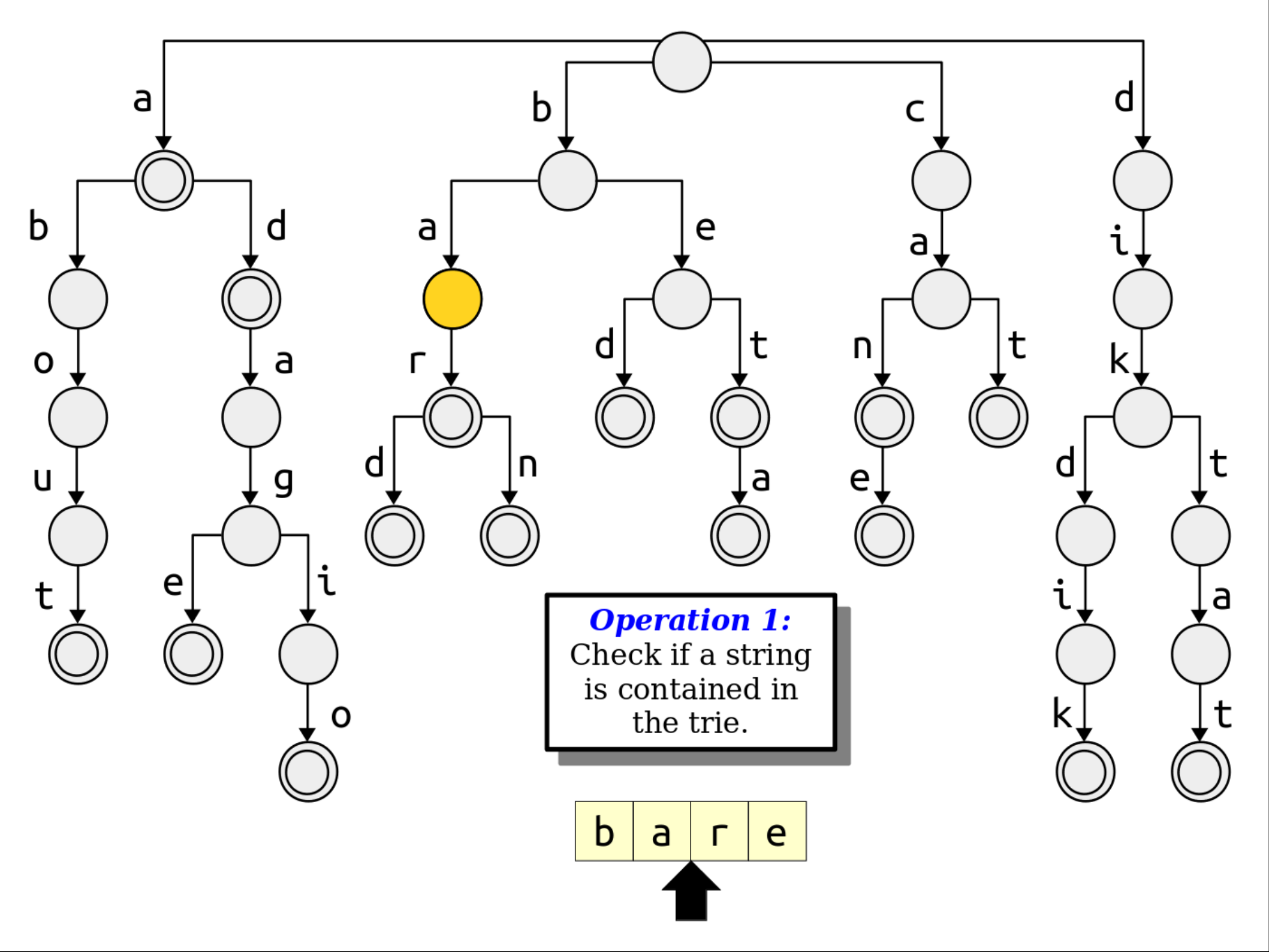


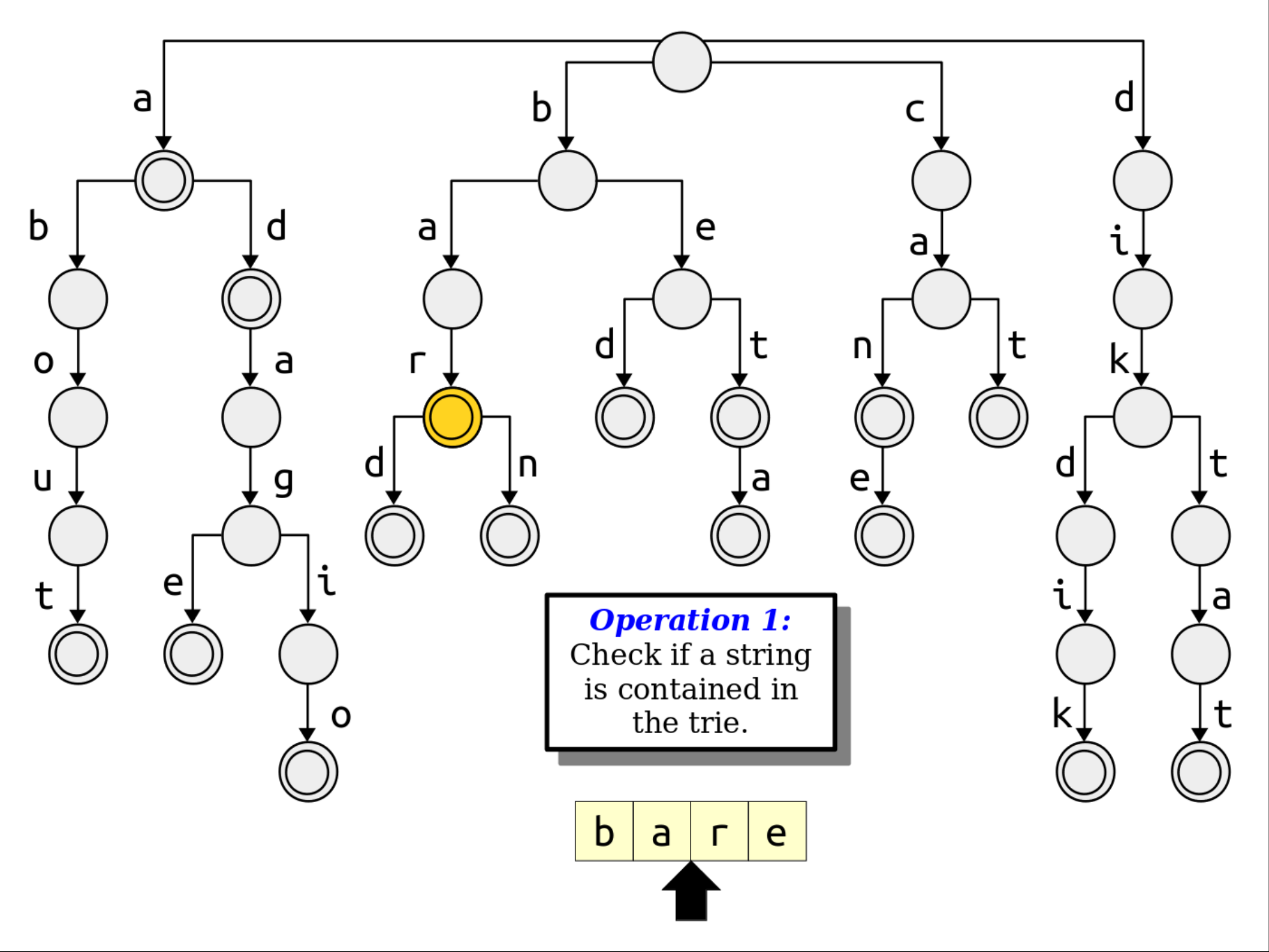
Operation 1:
Check if a string is contained in the trie.

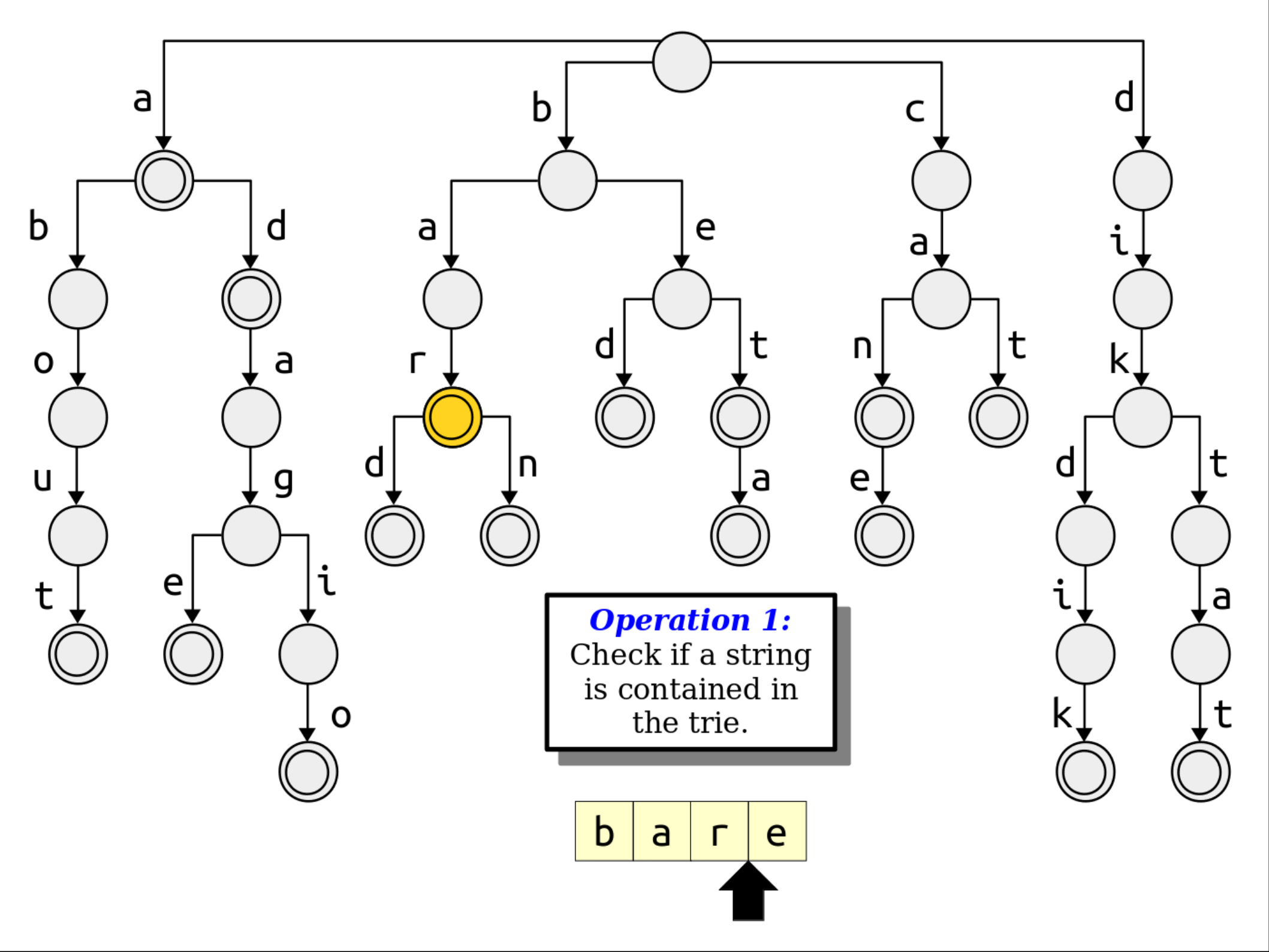
b	a	r	e


a
b
c
d
b
d
a
e
a
i
o
a
r
d
t
n
t
k
u
g
d
n
d
t
a
e
d
t
t
e
i
i
a
o
k
t
Operation 1:
Check if a string is contained in the trie.
b
a
r
e


a
b
c
d
b
d
a
e
a
i
o
a
r
d
t
n
t
k
u
g
d
n
d
t
a
e
d
t
t
e
i
i
a
o
k
t
Operation 1:
Check if a string is contained in the trie.
b
a
r
e

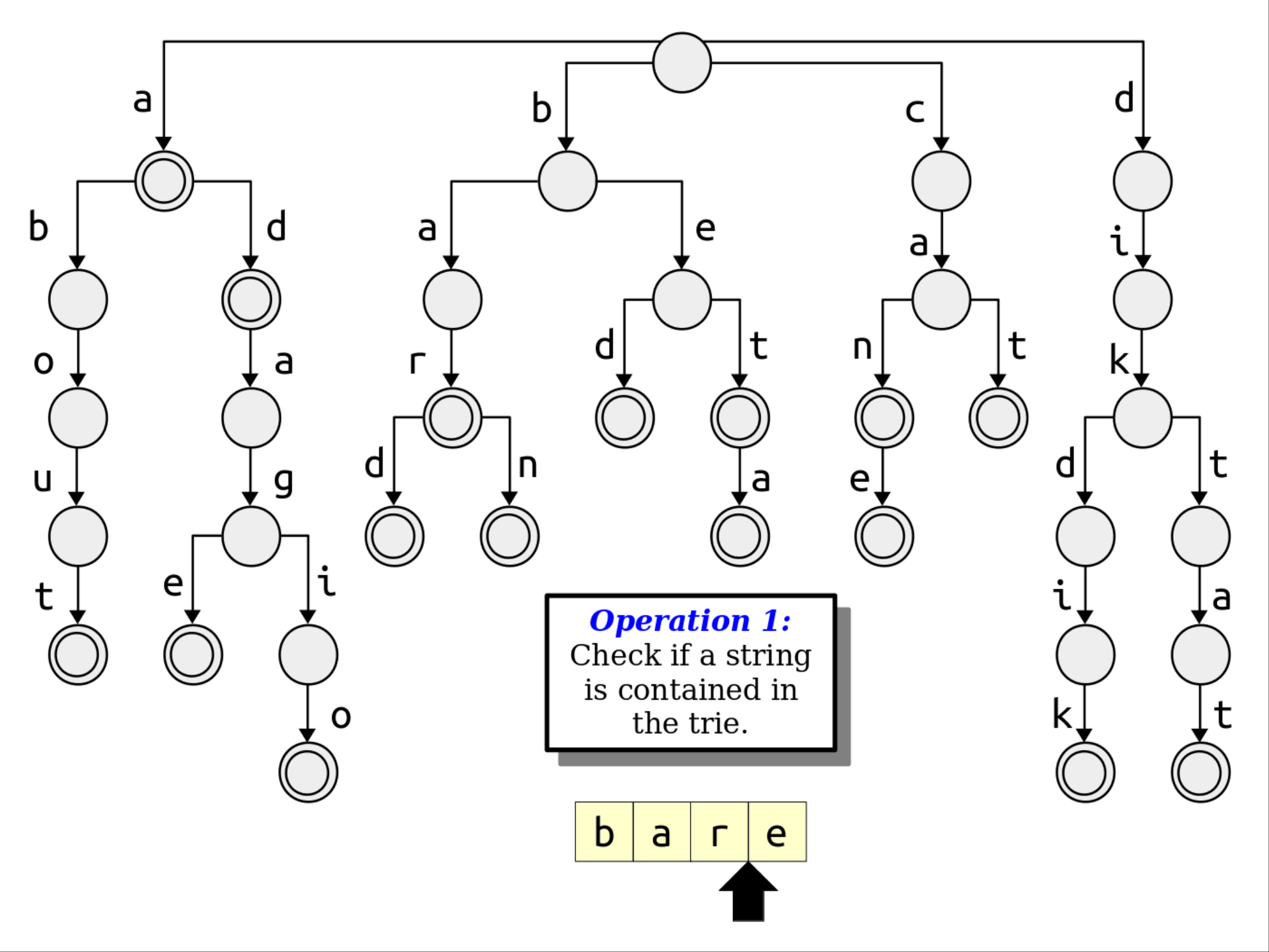

a
b
d
o
a
u
g
t
e
i
o
b
a
e
r
d
t
d
n
a
a
c
n
t
e
d
i
k
d
t
i
a
k
t
Operation 1:
Check if a string is contained in the trie.
b
a
r
e


a
b
c
d
b
d
a
e
a
i
o
a
r
d
t
n
t
k
u
g
d
n
d
t
a
e
d
t
t
e
i
i
a
o
k
t
Operation 1:
Check if a string is contained in the trie.
b
a
r
e



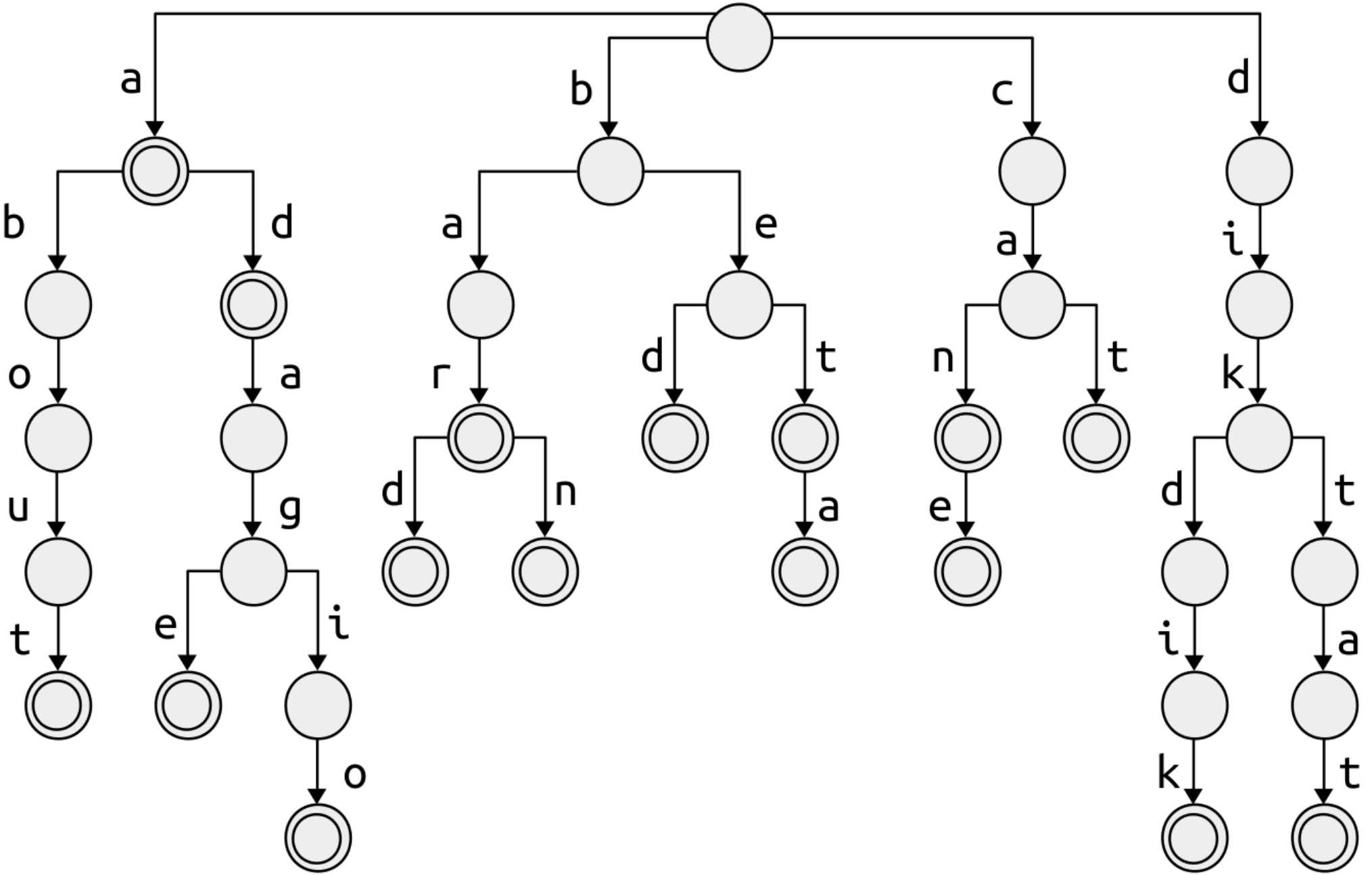
Operation 1:
Check if a string is contained in the trie.

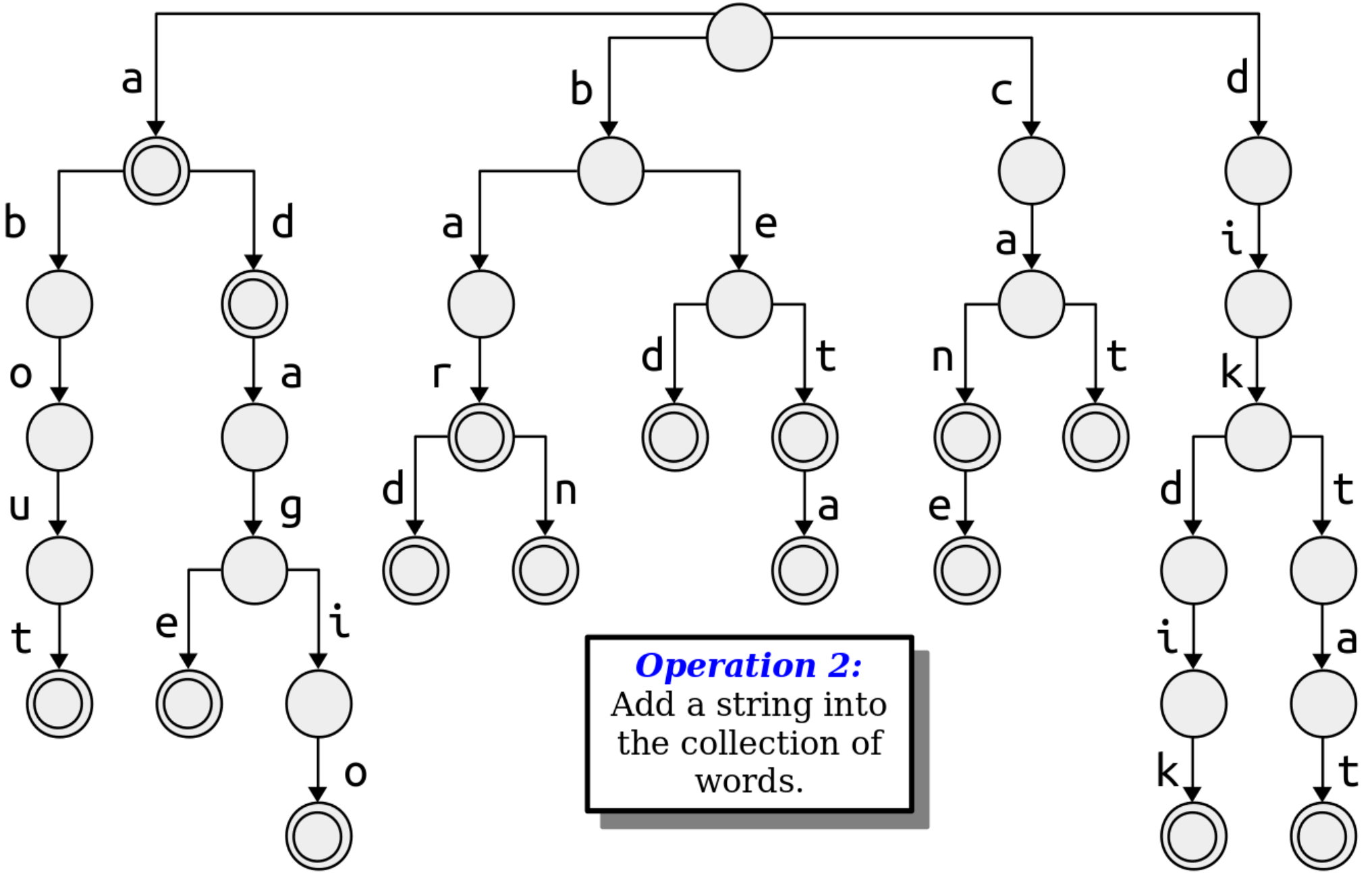
b	a	r	e



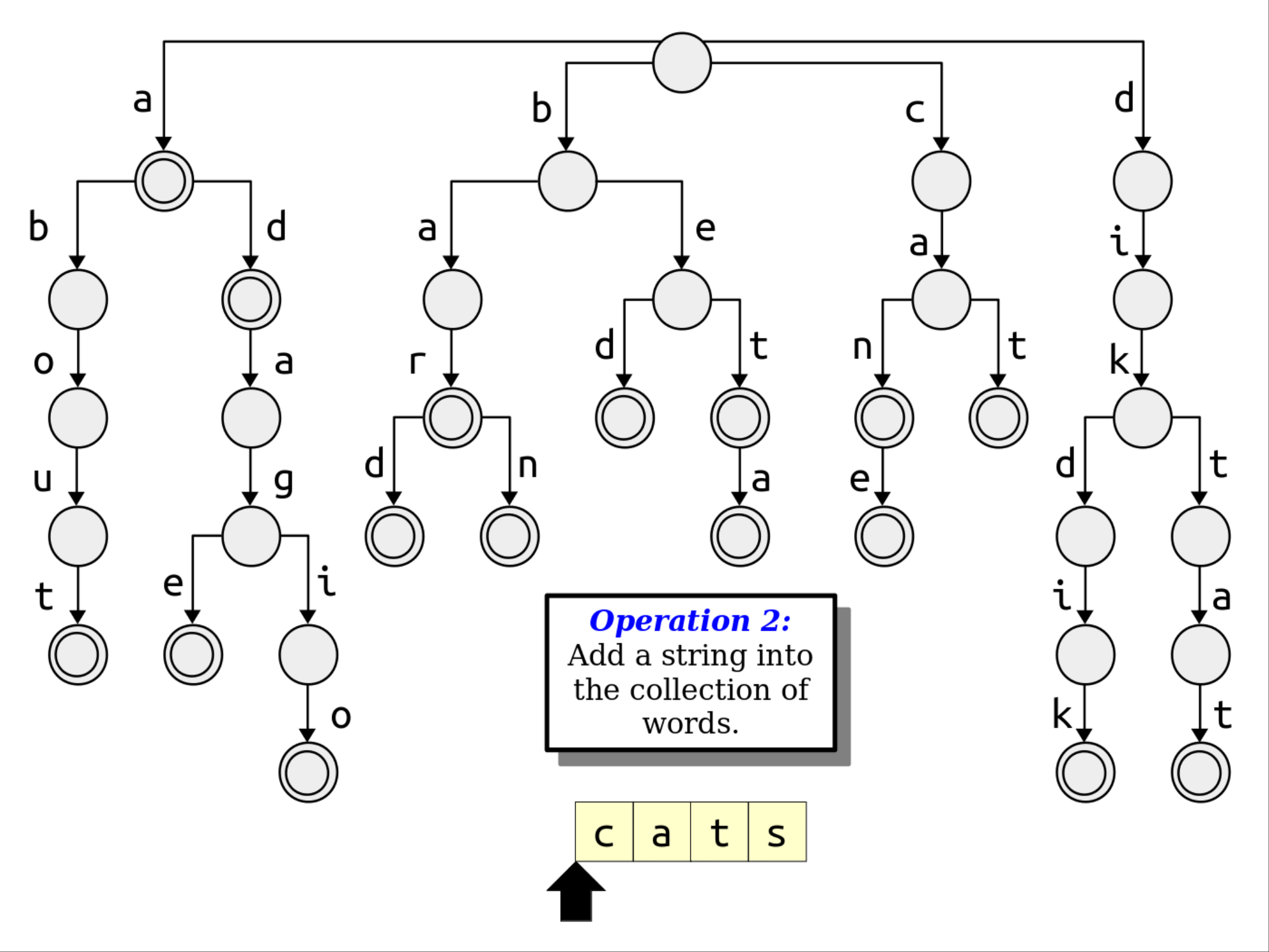
Operation 1:
Check if a string is contained in the trie.

b	a	r	e


a
b
c
d
b
d
a
e
a
i
o
a
r
d
t
n
t
k
u
g
d
n
d
a
e
d
t
t
e
i
i
a
o
k
t

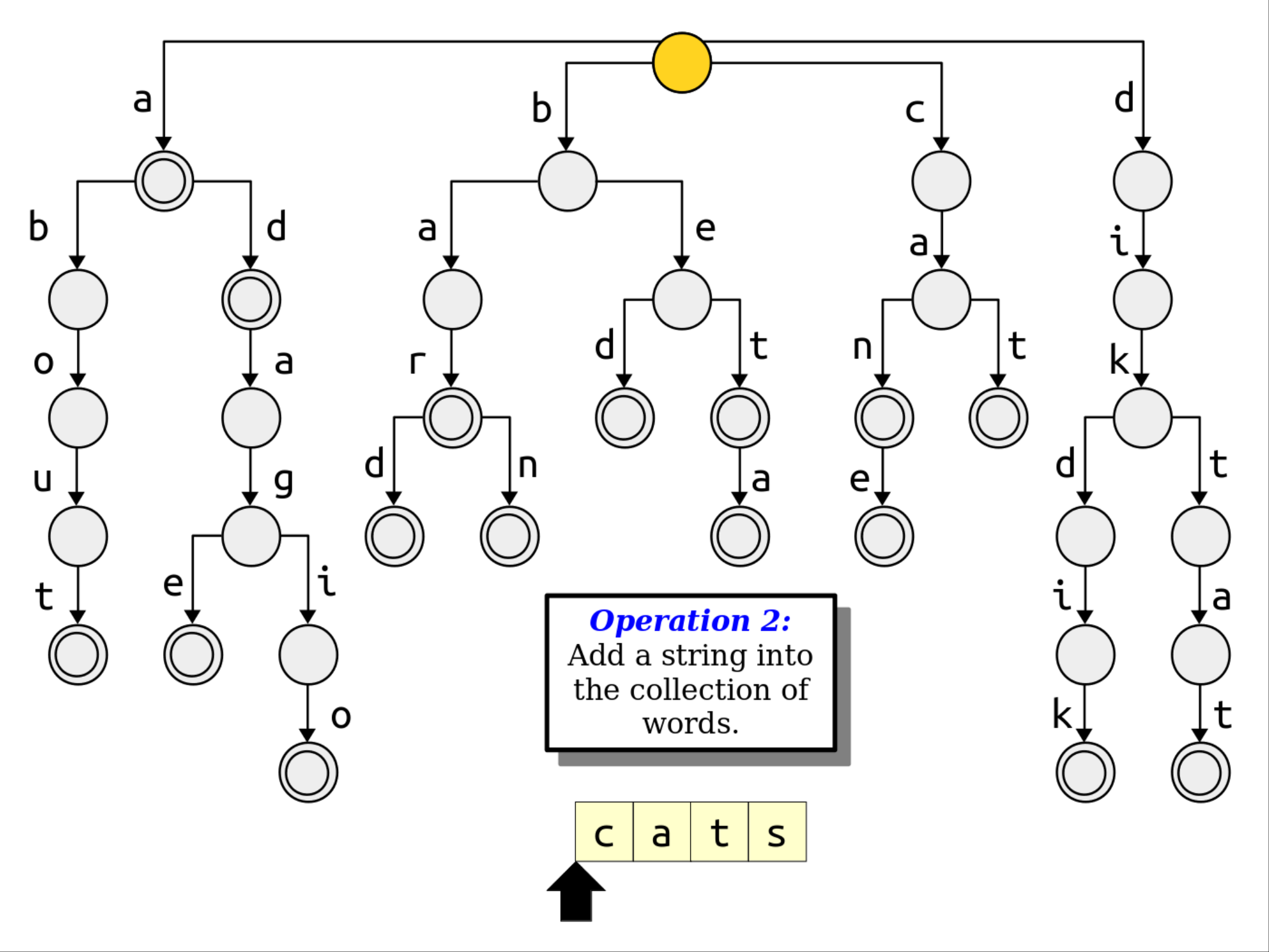


Operation 2:
Add a string into the collection of words.



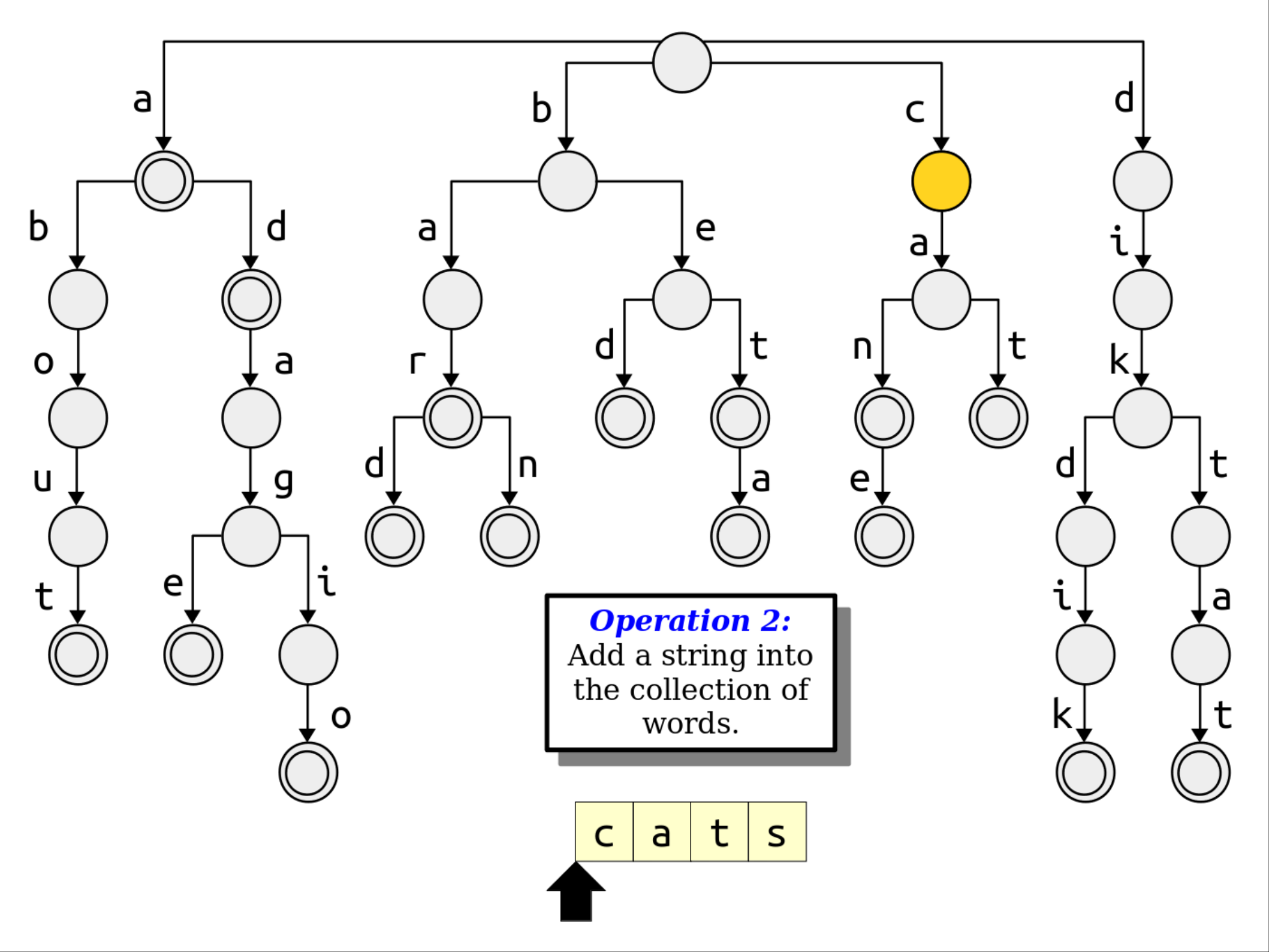
Operation 2:
Add a string into the collection of words.

c	a	t	s



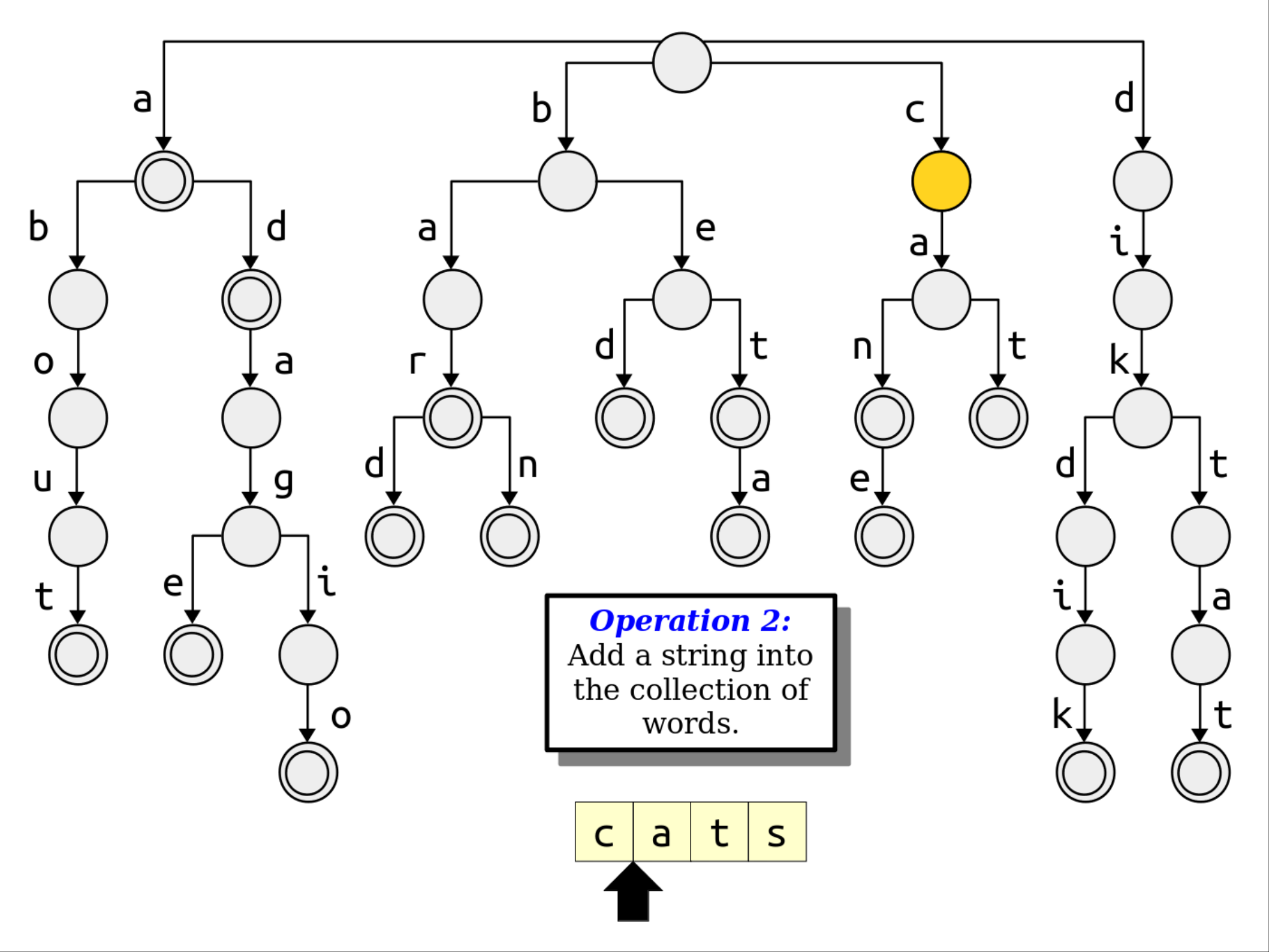
Operation 2:
Add a string into the collection of words.

c	a	t	s



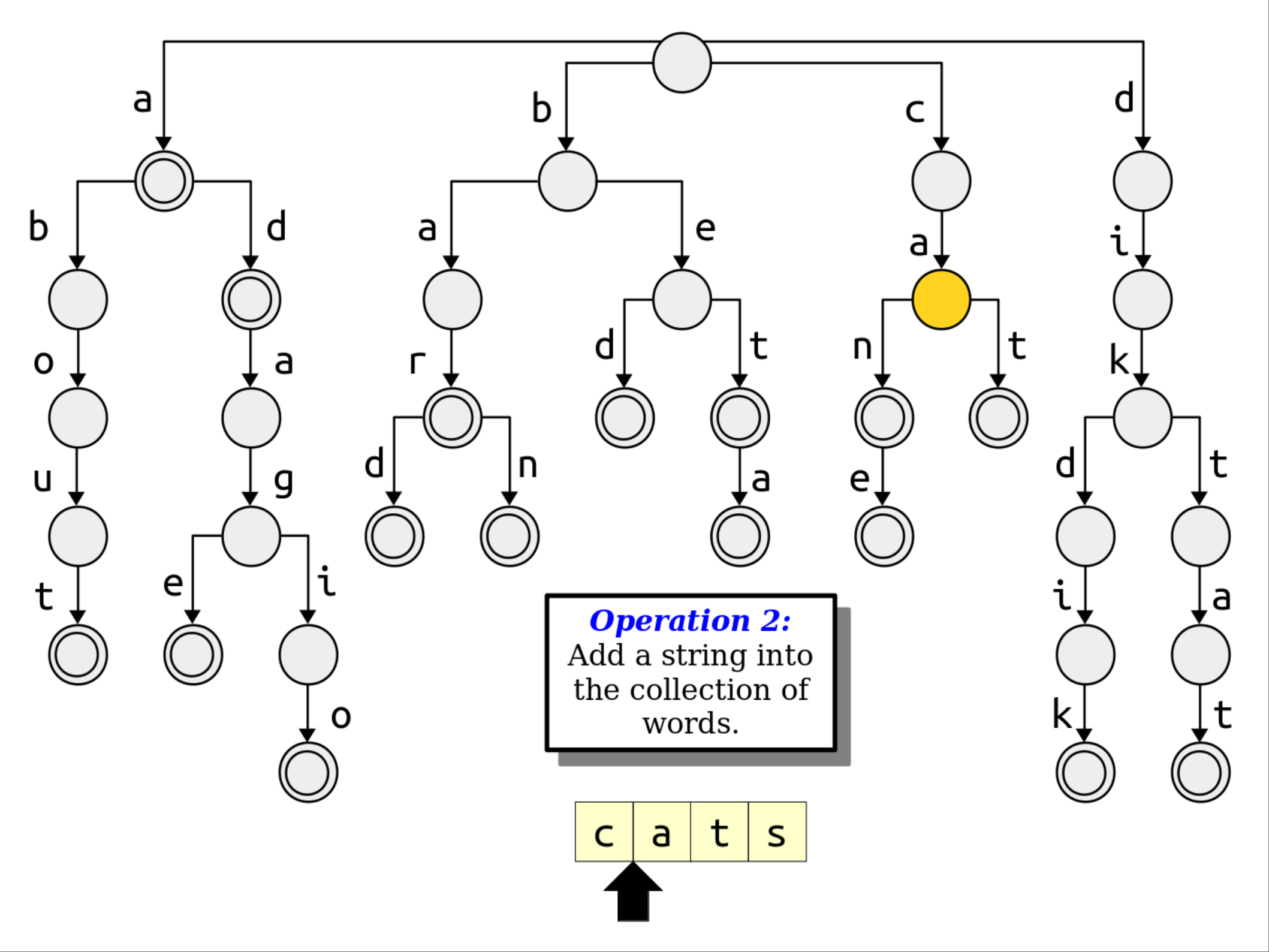
Operation 2:
Add a string into the collection of words.

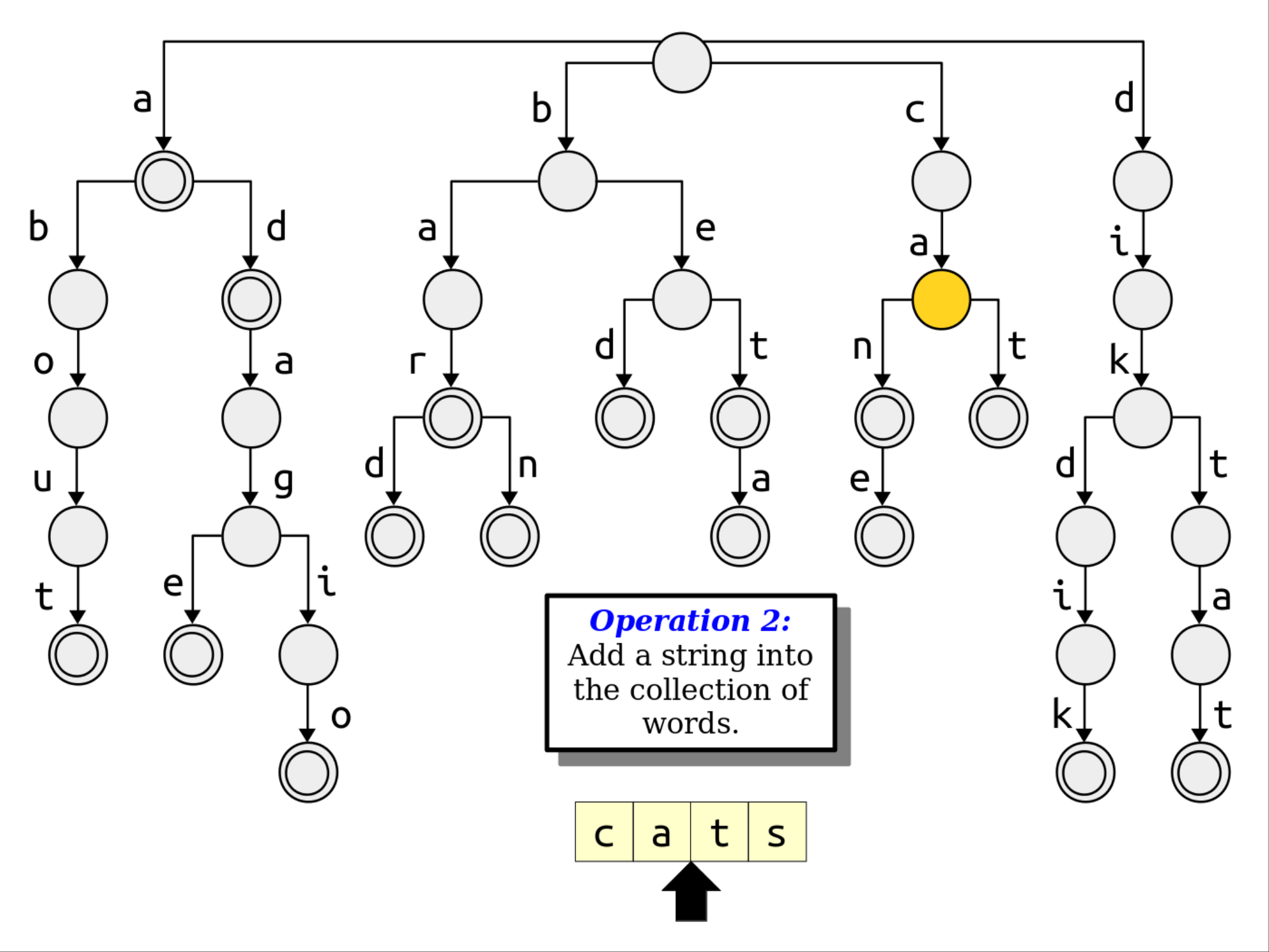
c	a	t	s



Operation 2:
Add a string into the collection of words.

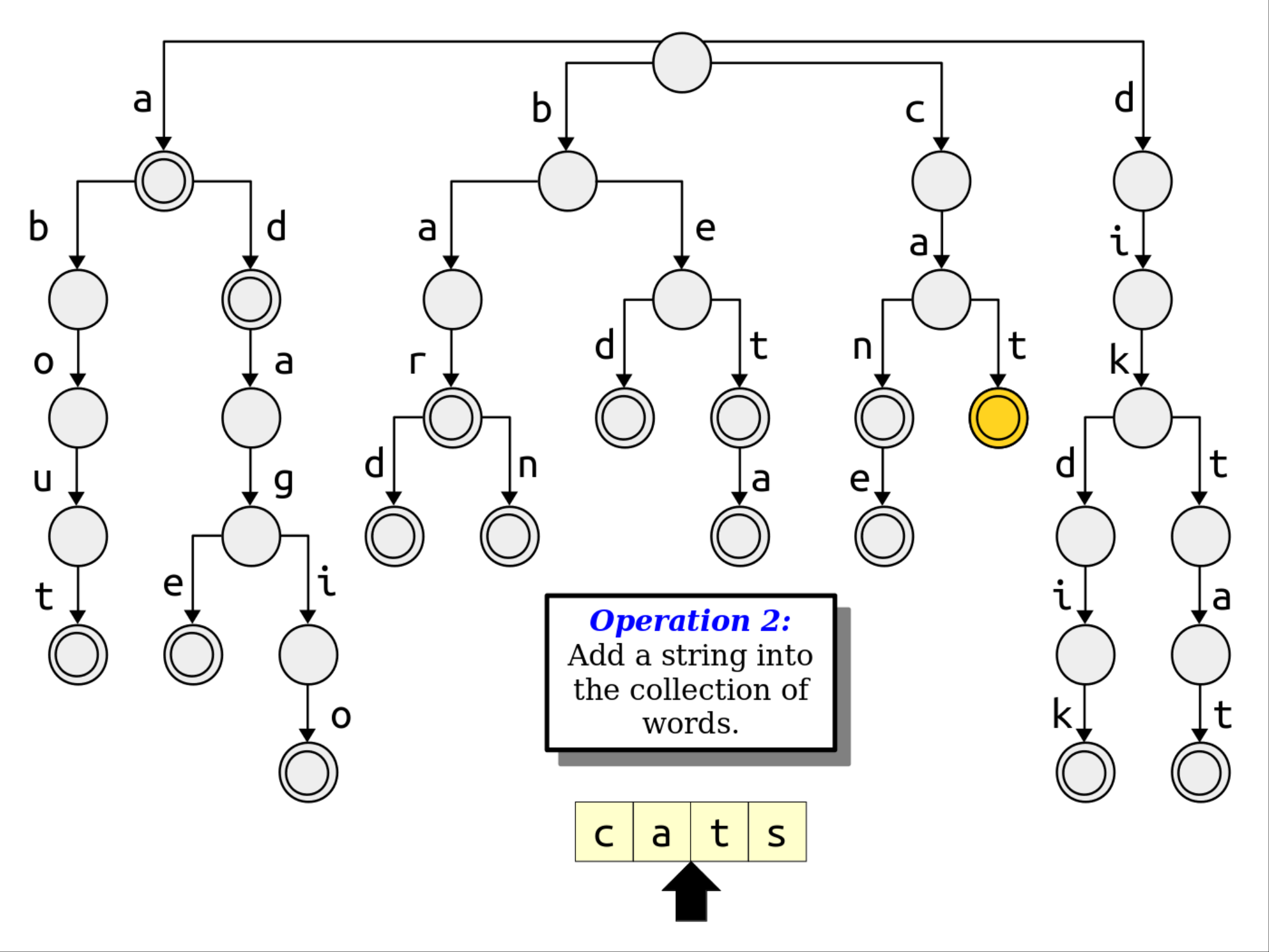
c	a	t	s


a
b
c
d
b
d
a
e
a
i
o
a
r
d
t
n
t
k
u
g
d
n
d
a
e
d
t
t
e
i
i
a
o
k
t
Operation 2:
Add a string into the collection of words.
c
a
t
s



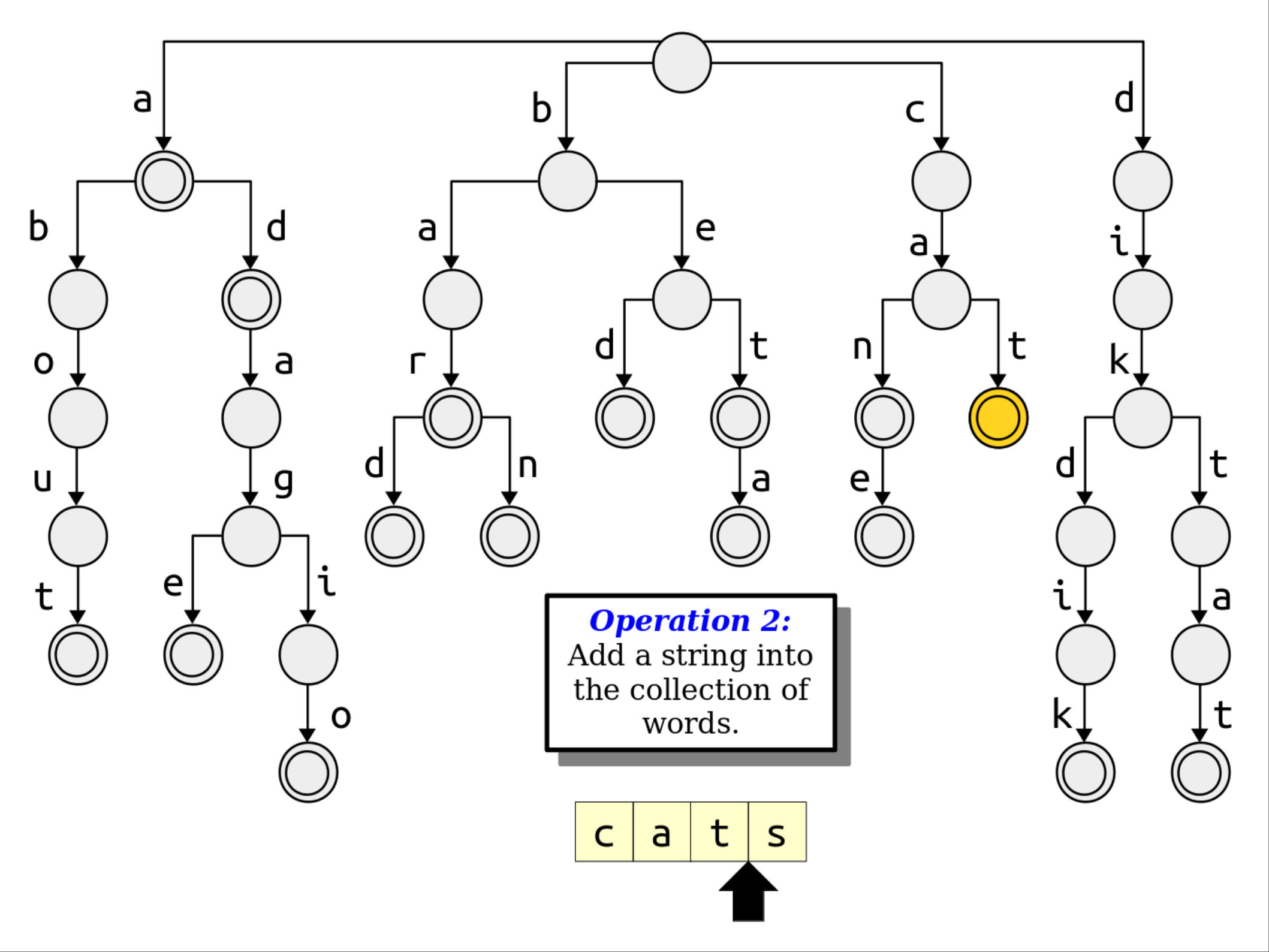
Operation 2:
Add a string into the collection of words.

c	a	t	s



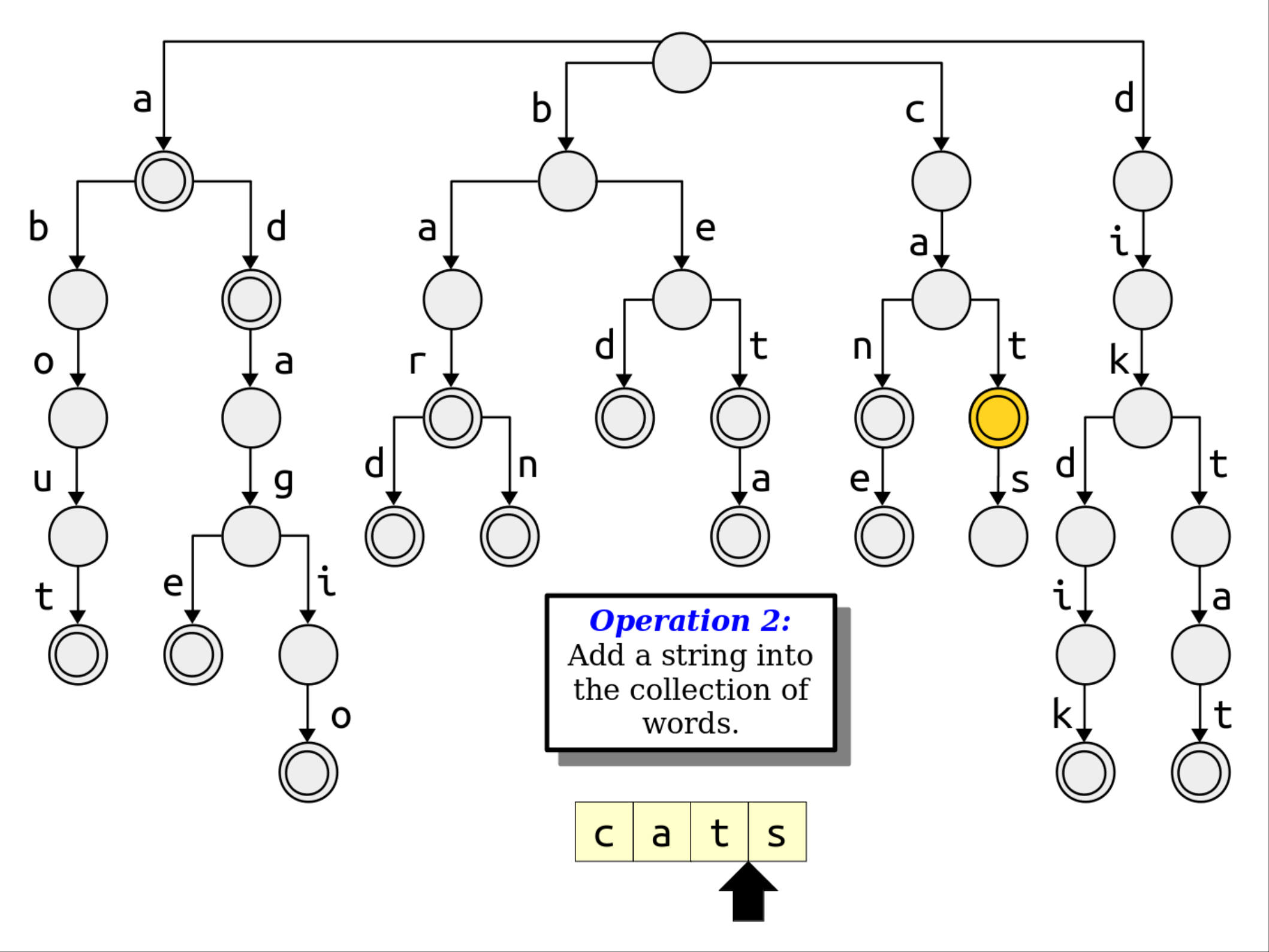
Operation 2:
Add a string into the collection of words.

c	a	t	s



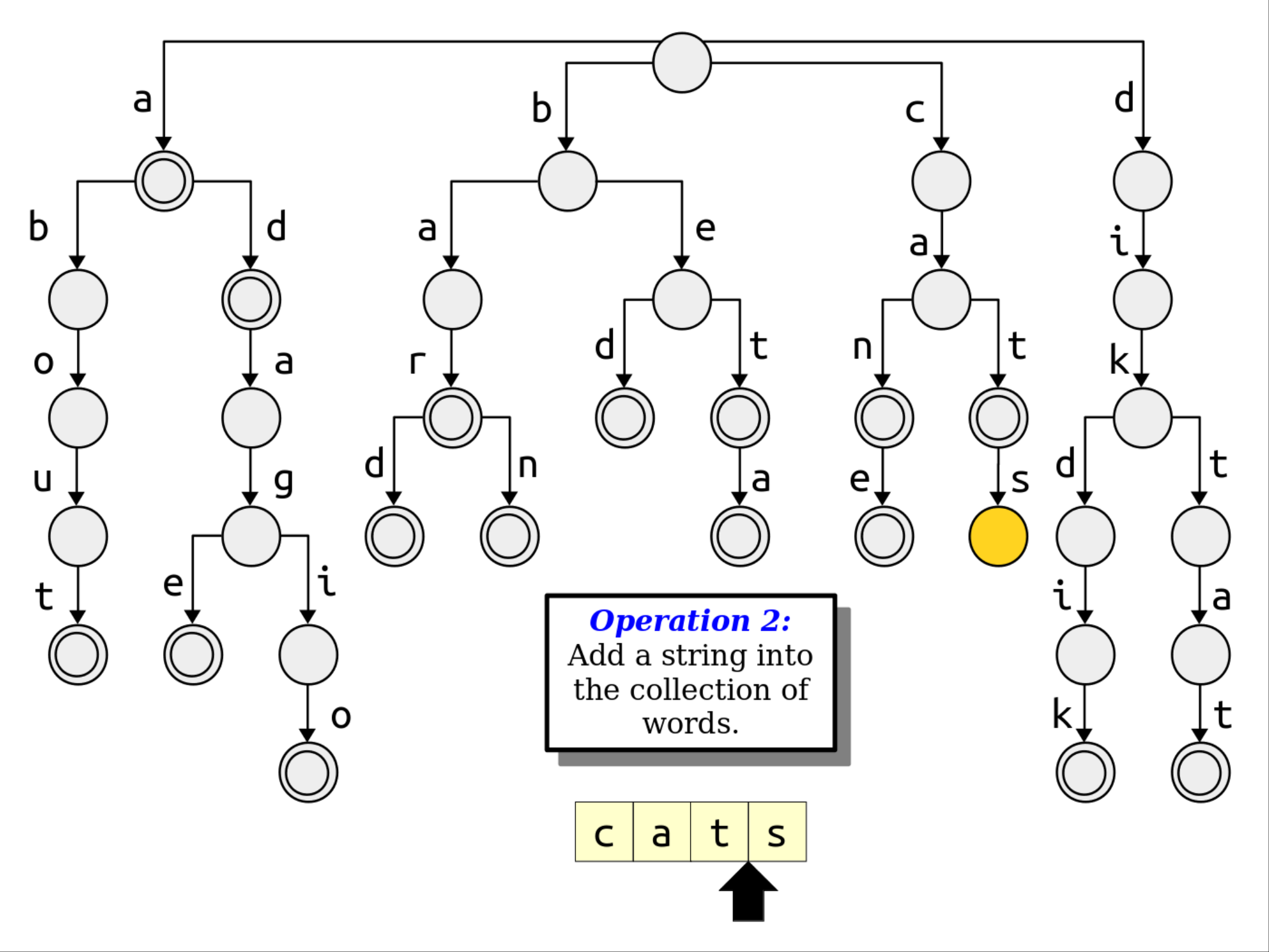
Operation 2:
Add a string into the collection of words.

c	a	t	s



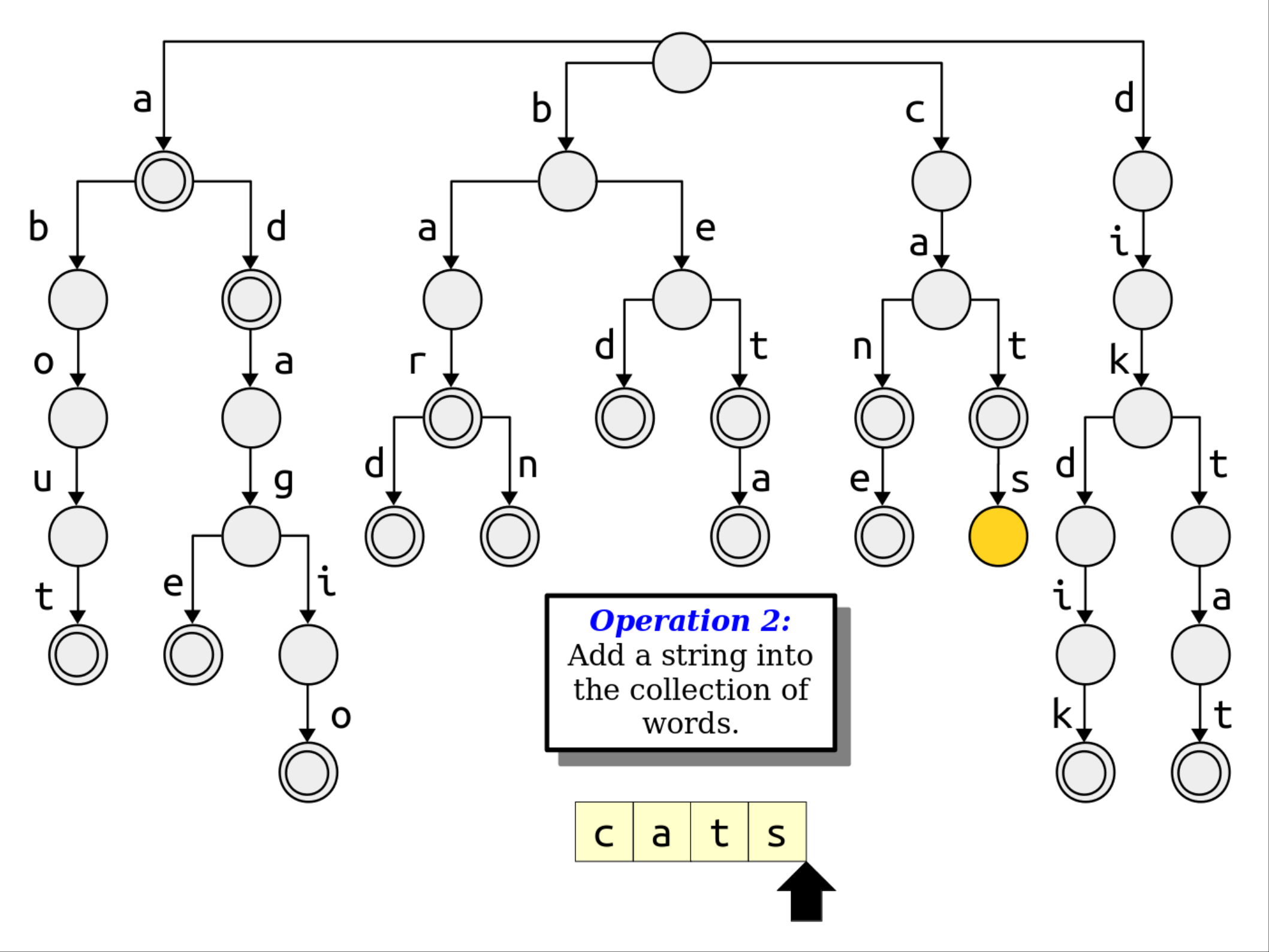
Operation 2:
Add a string into the collection of words.

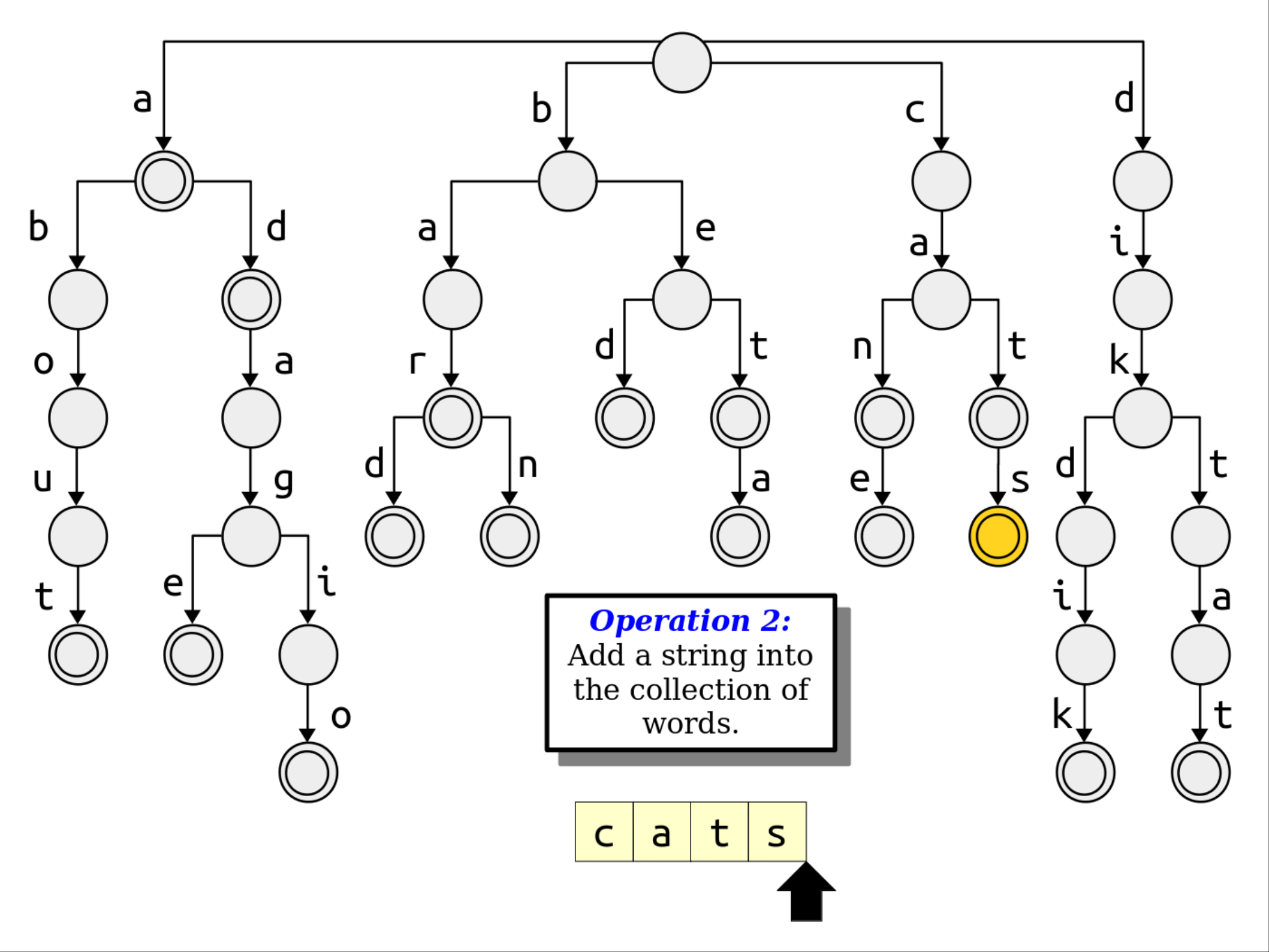
c	a	t	s



Operation 2:
Add a string into the collection of words.

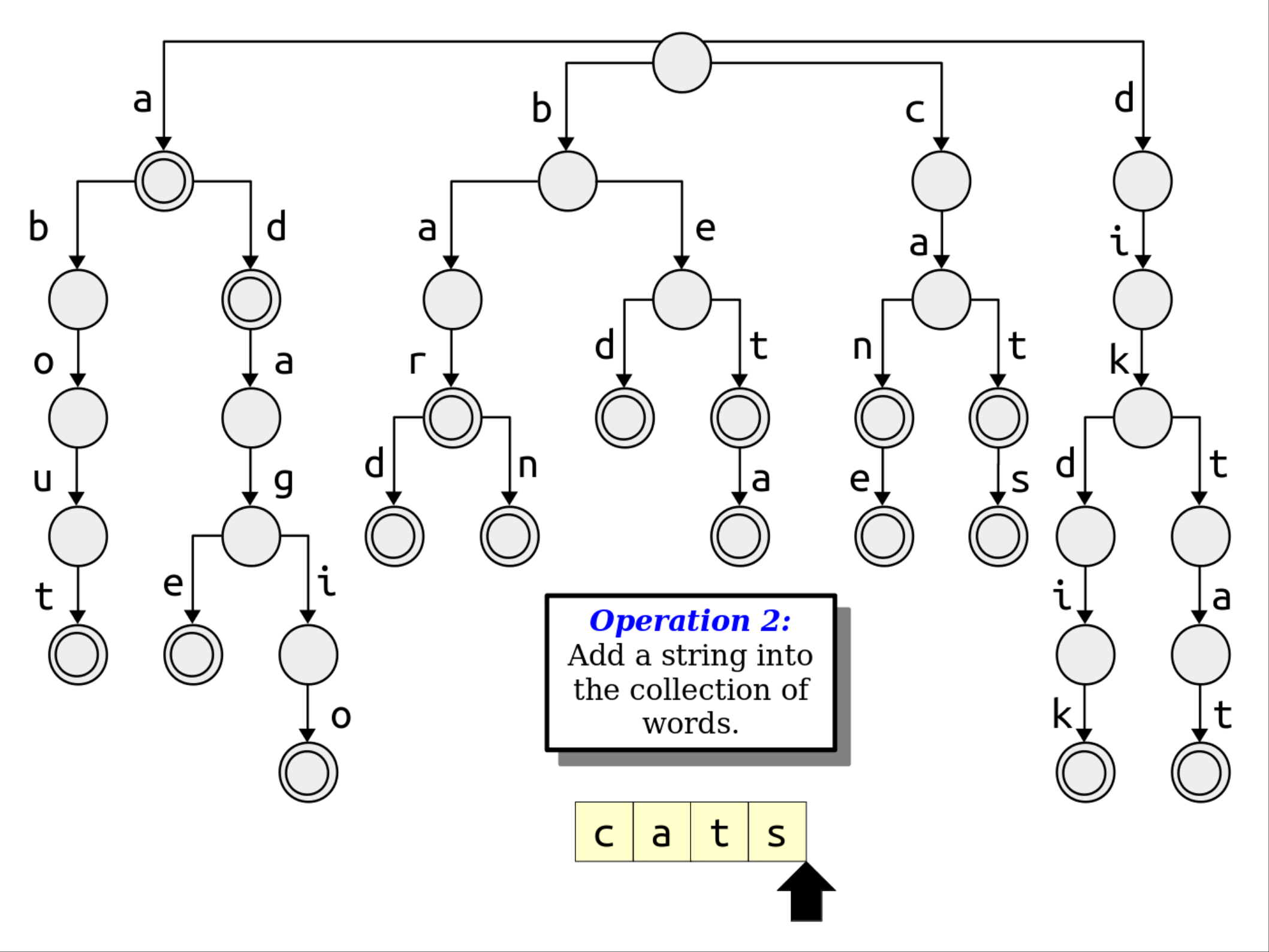
c	a	t	s


a
b
c
d
b
d
a
e
a
i
o
a
r
d
t
n
t
k
u
g
d
n
d
t
e
s
t
e
i
i
a
o
k
t
Operation 2:
Add a string into the collection of words.
c
a
t
s



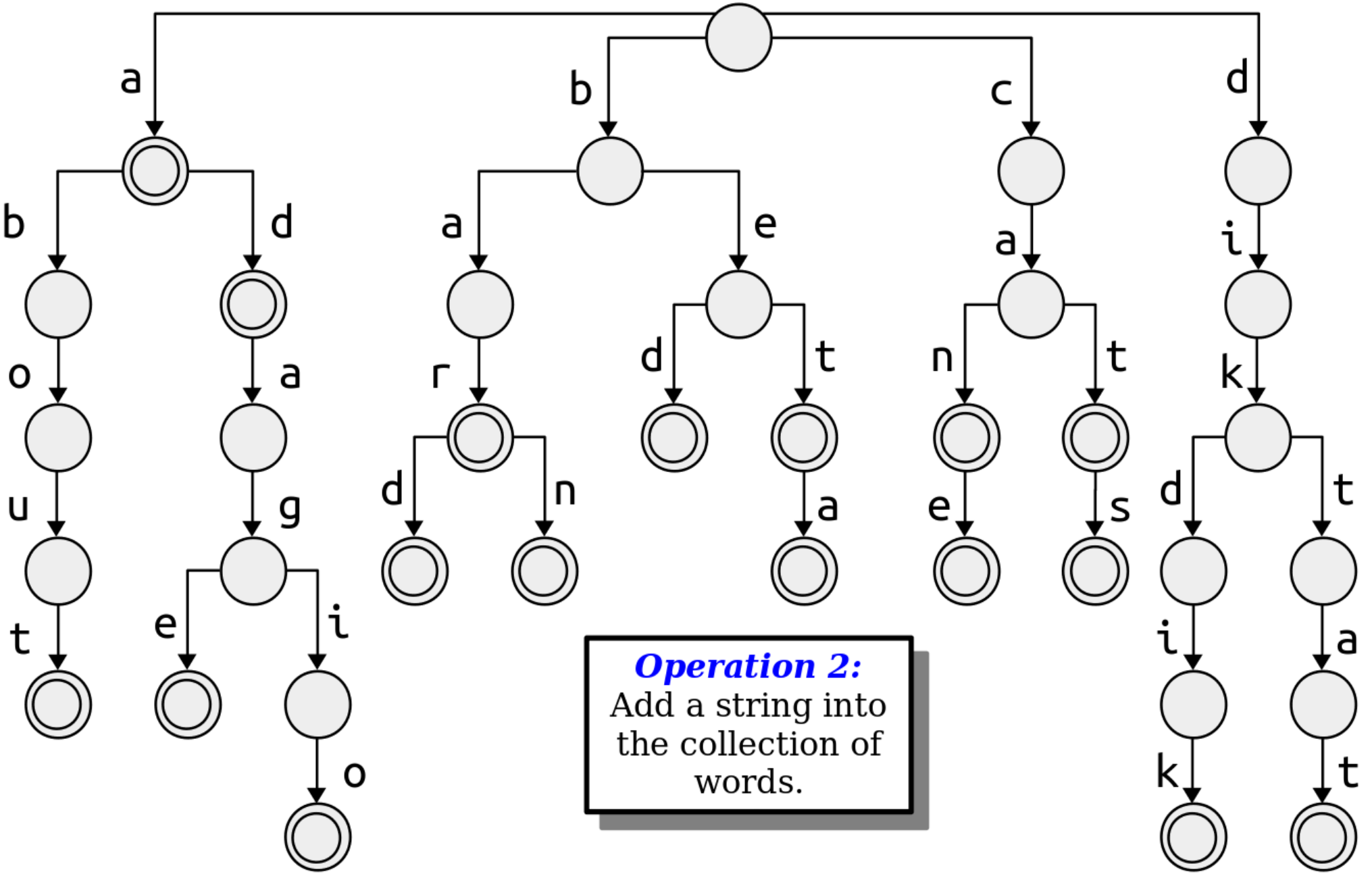
Operation 2:
Add a string into the collection of words.

c	a	t	s

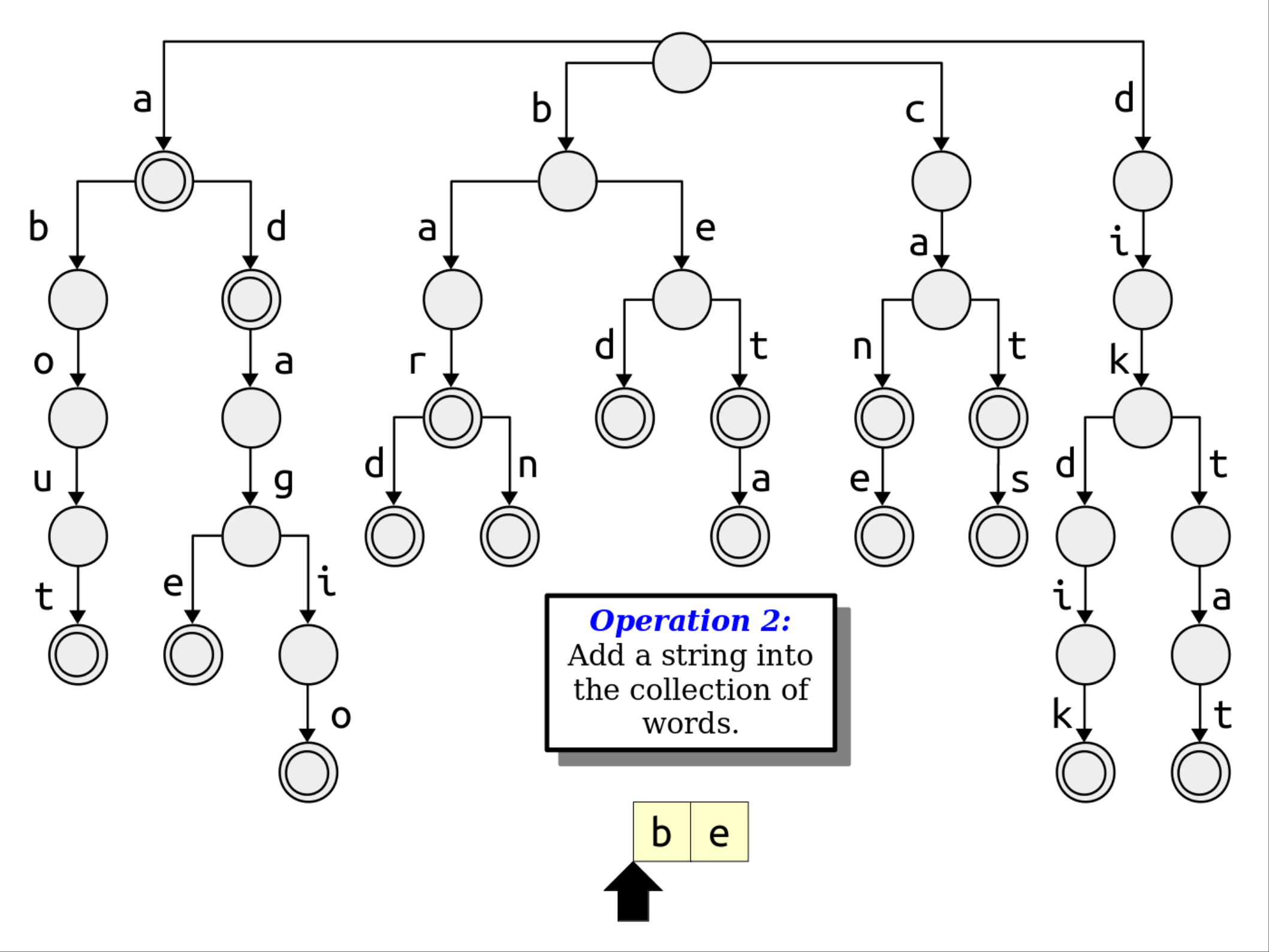


Operation 2:
Add a string into the collection of words.

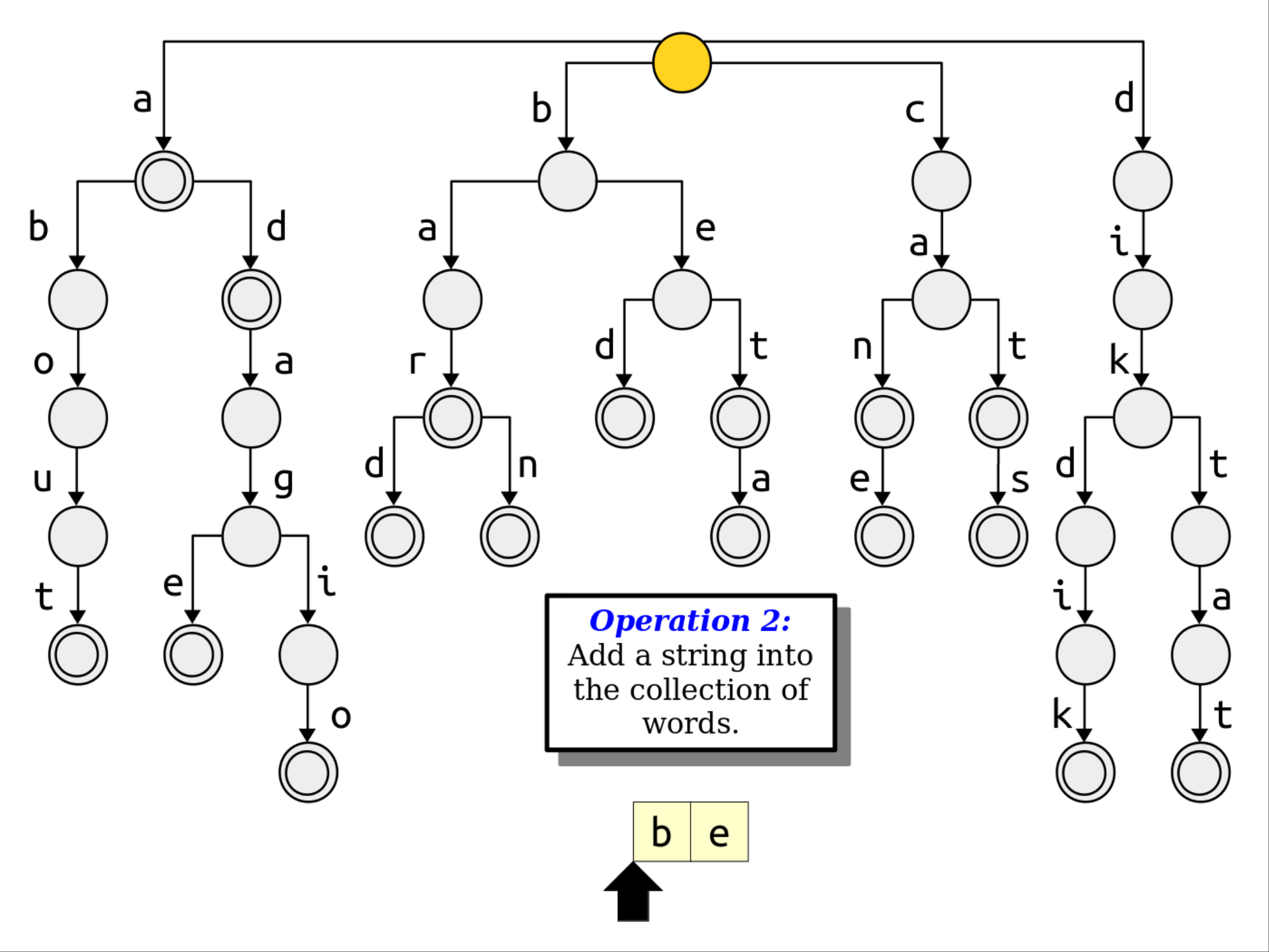
c	a	t	s

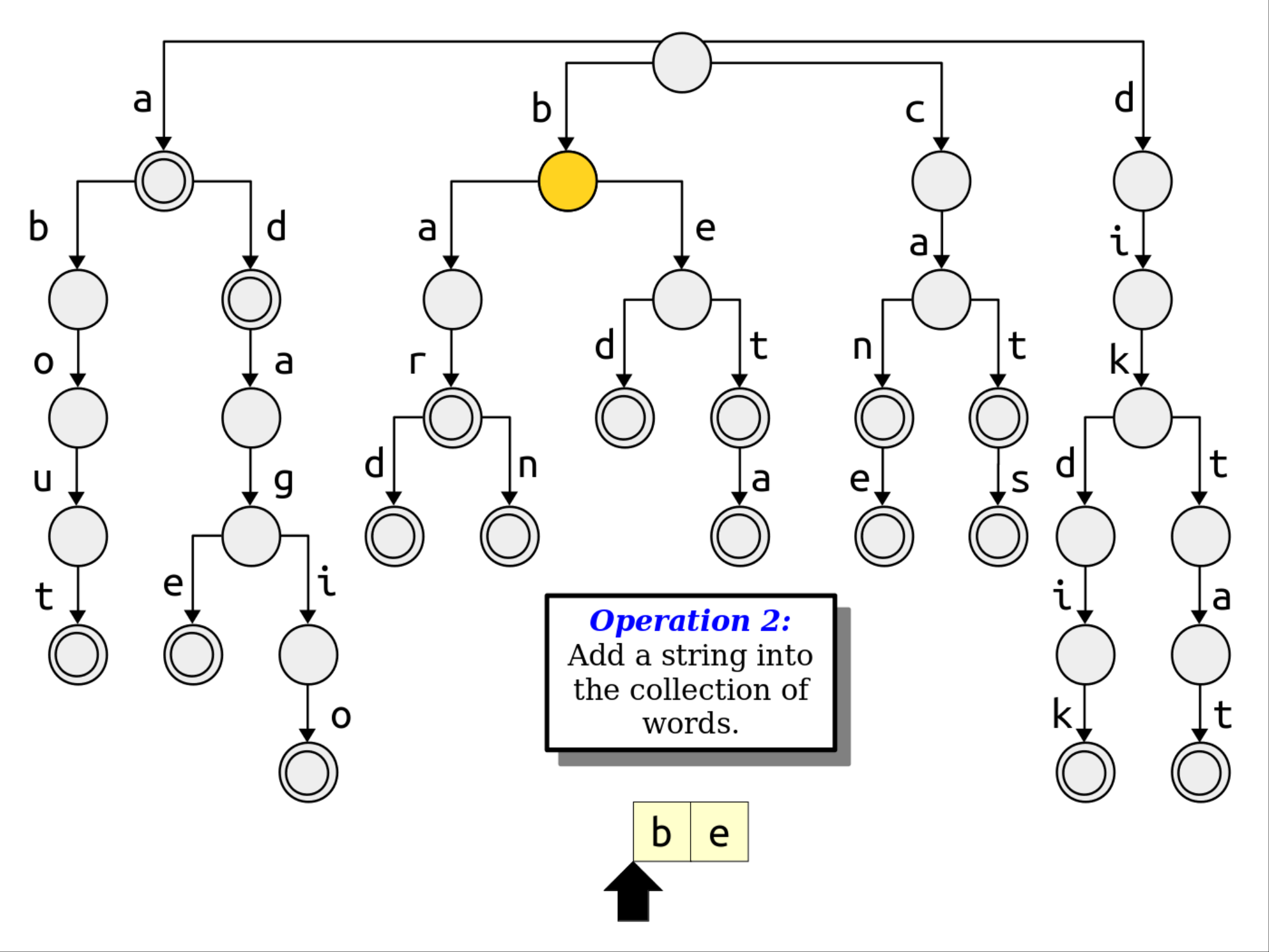


Operation 2:
Add a string into the collection of words.

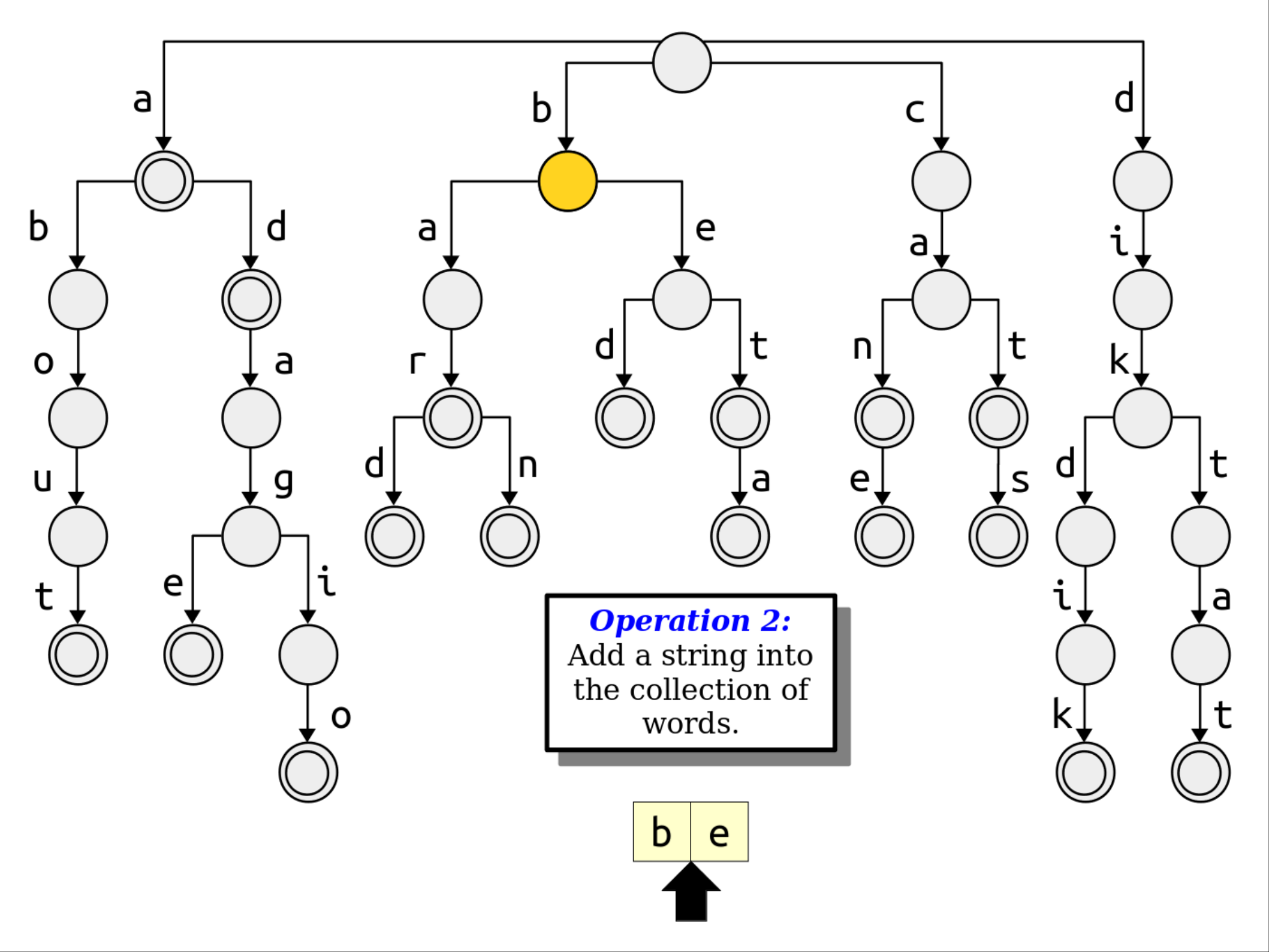


Operation 2:
Add a string into the collection of words.

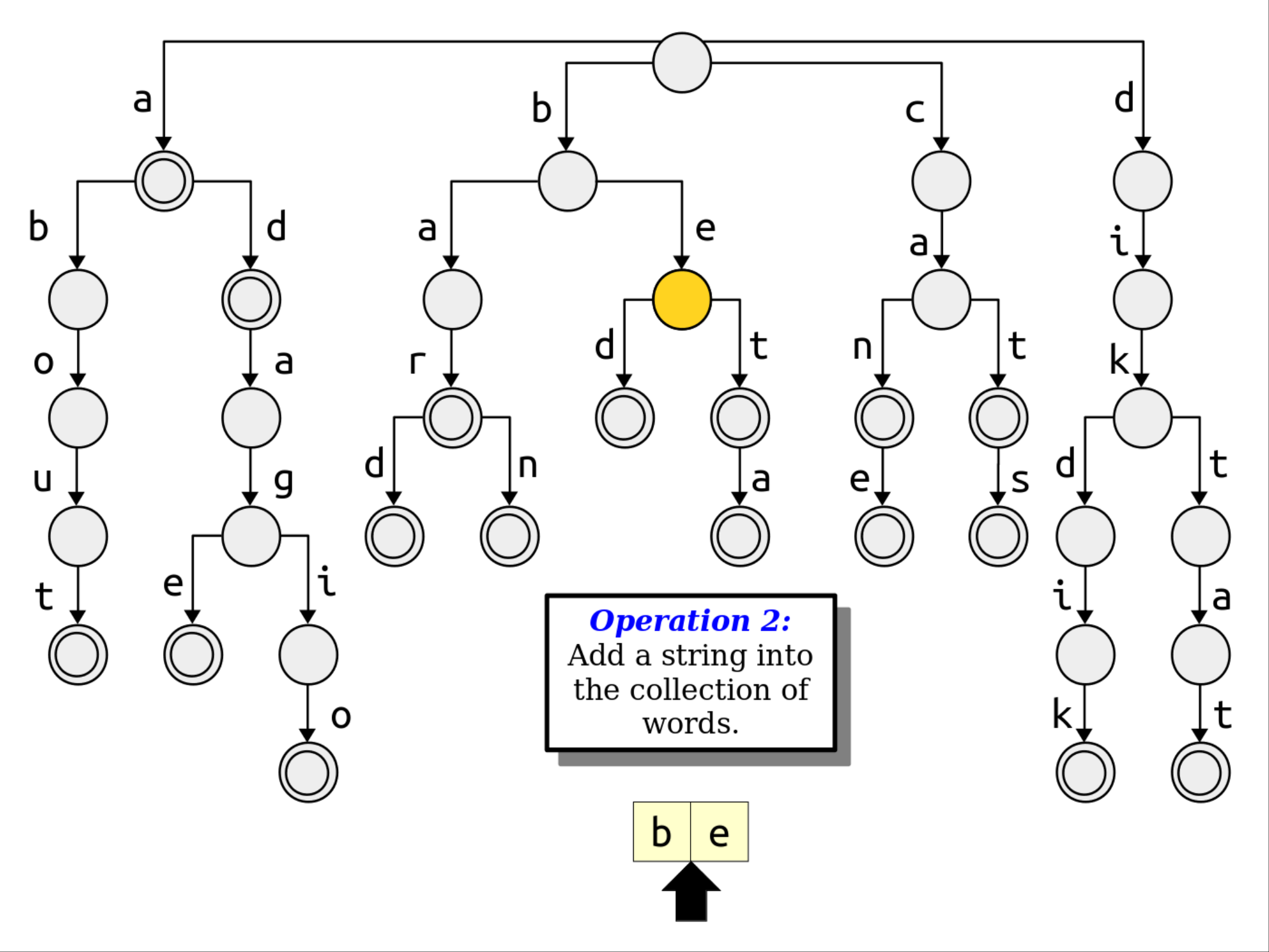

a
b
c
d
b
d
a
e
a
i
o
a
r
d
t
n
t
k
u
g
d
n
d
t
e
s
t
e
i
i
a
o
k
t
Operation 2:
Add a string into the collection of words.
b
e



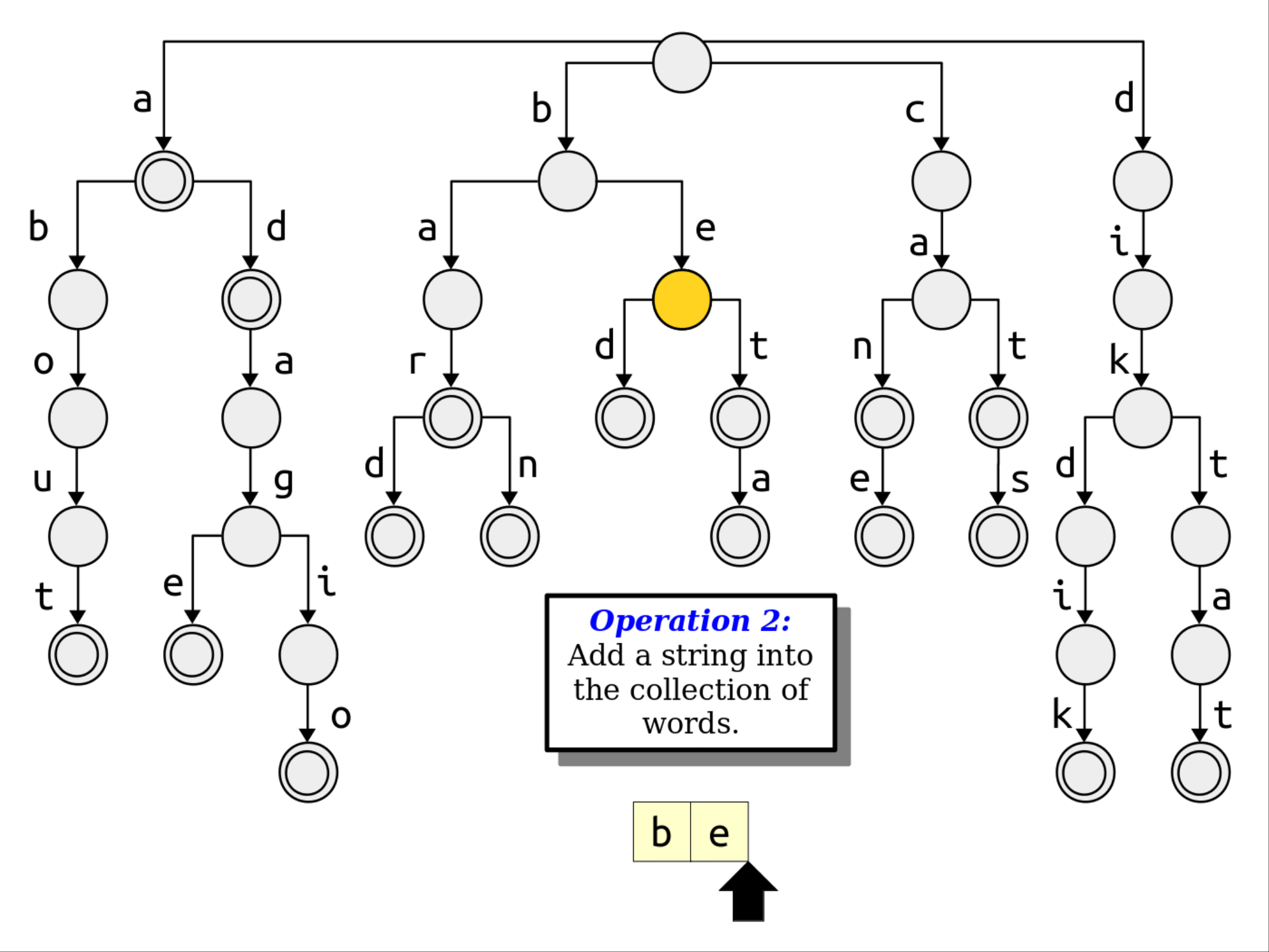
Operation 2:
Add a string into the collection of words.



Operation 2:
Add a string into the collection of words.

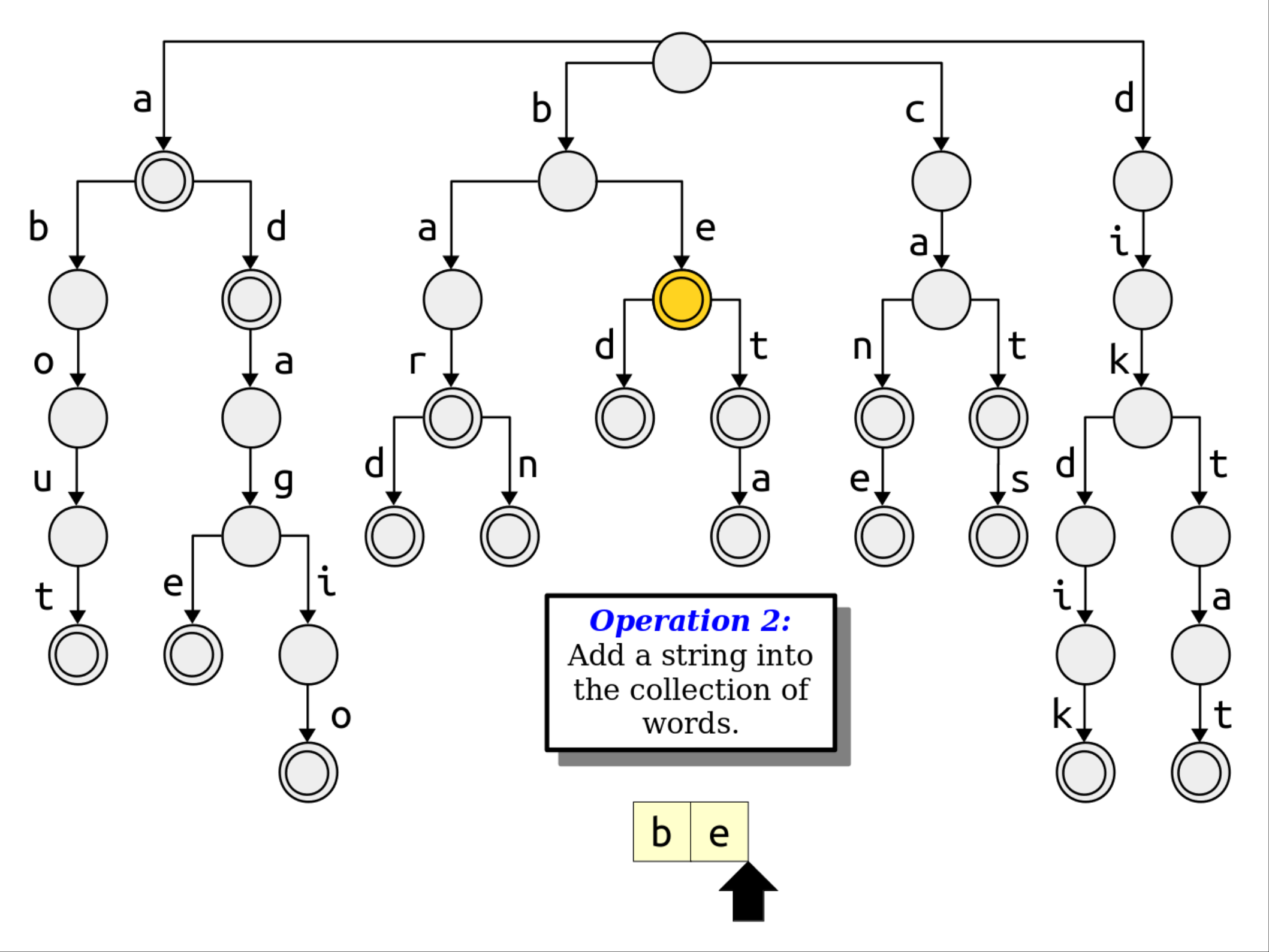


Operation 2:
Add a string into the collection of words.



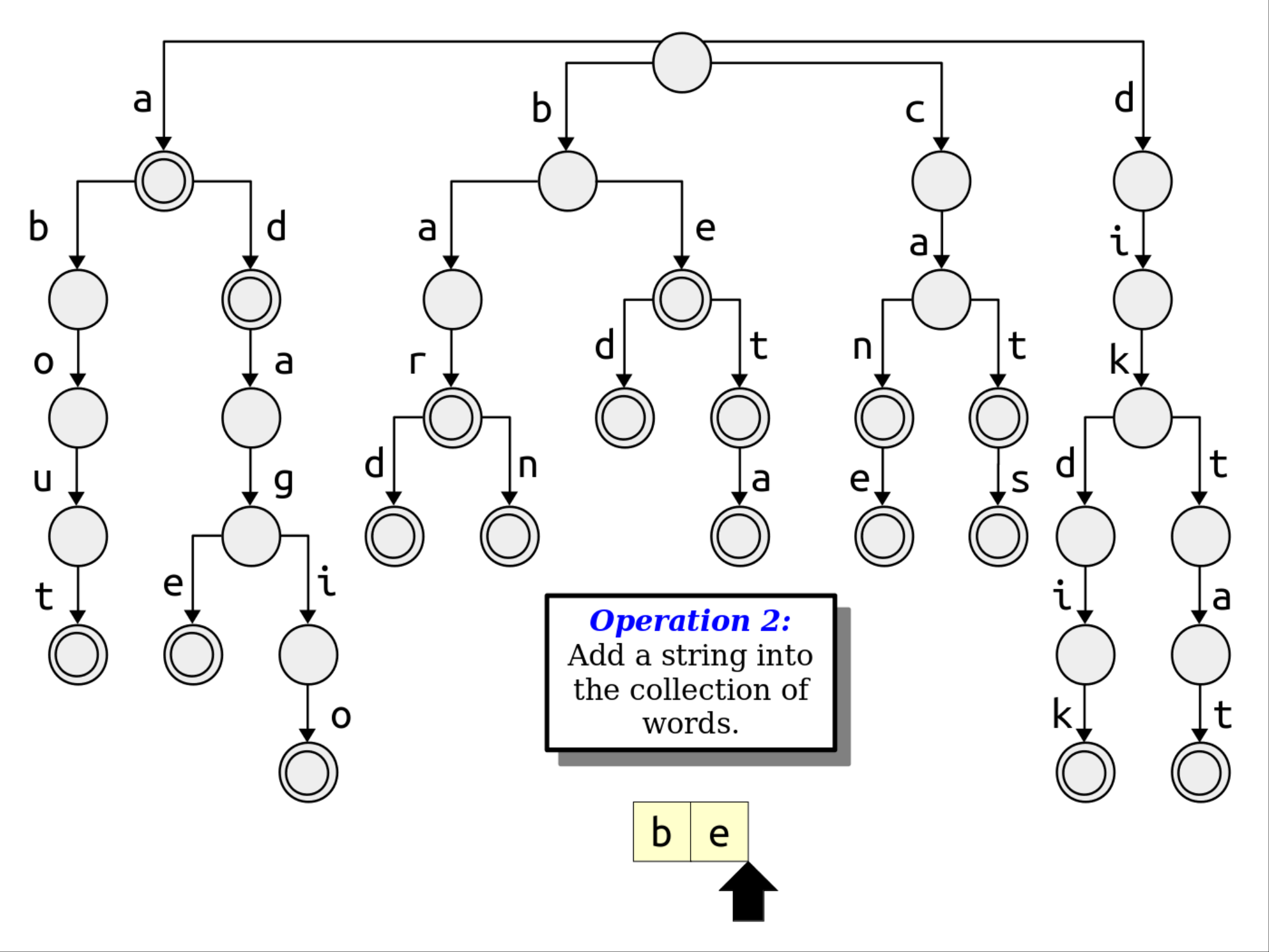
Operation 2:
Add a string into the collection of words.

b e



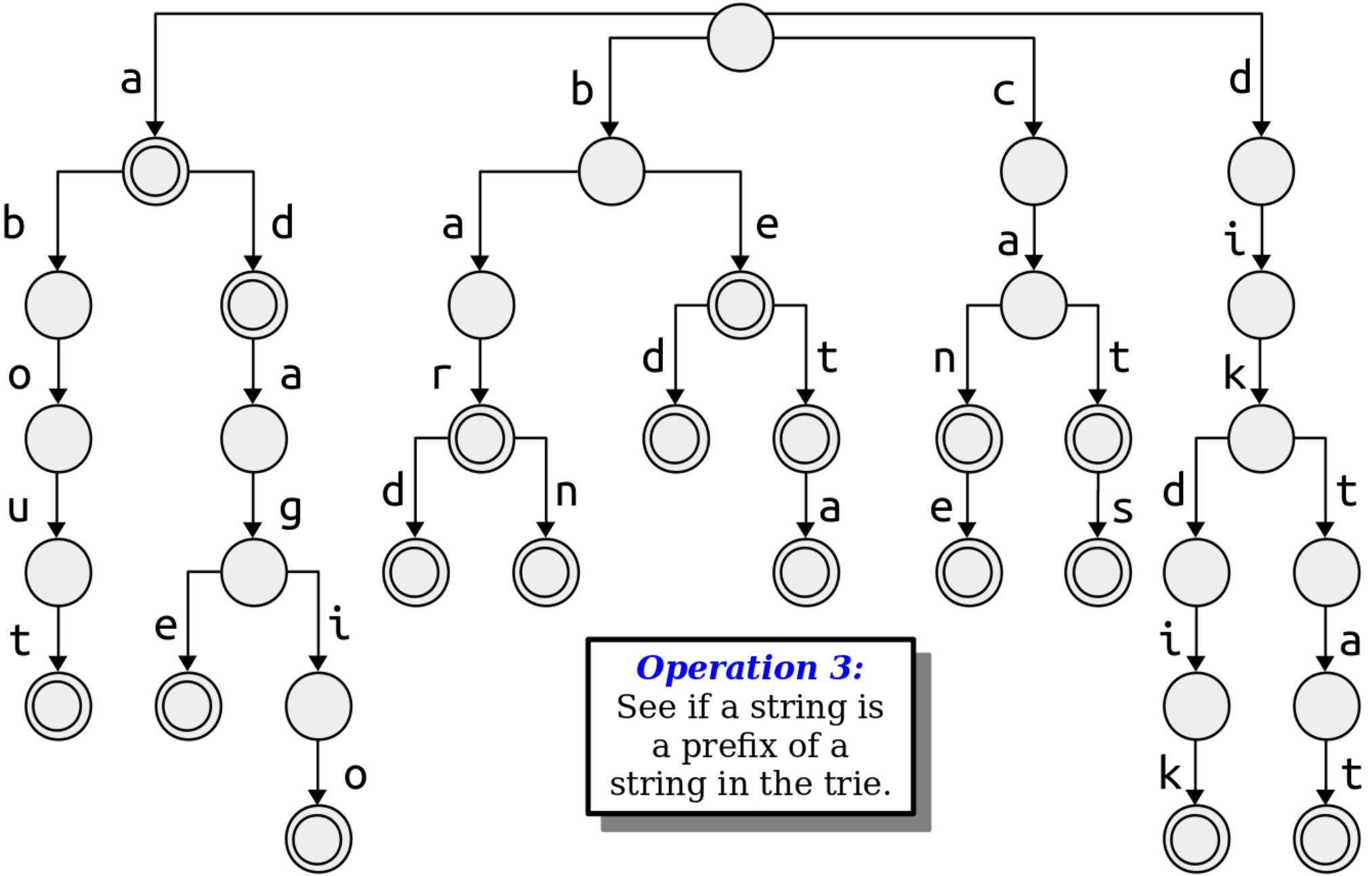
Operation 2:
Add a string into the collection of words.

b e

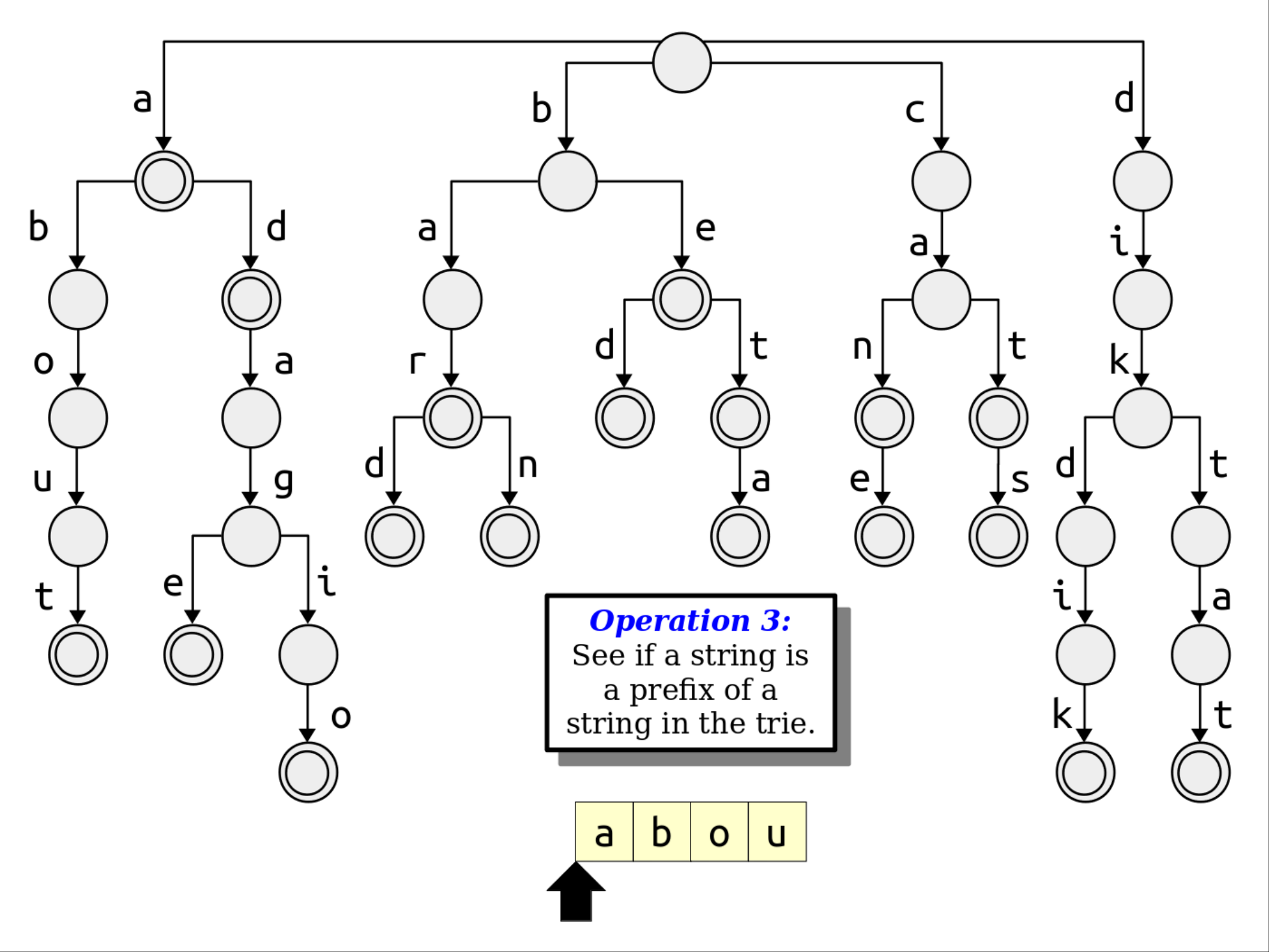


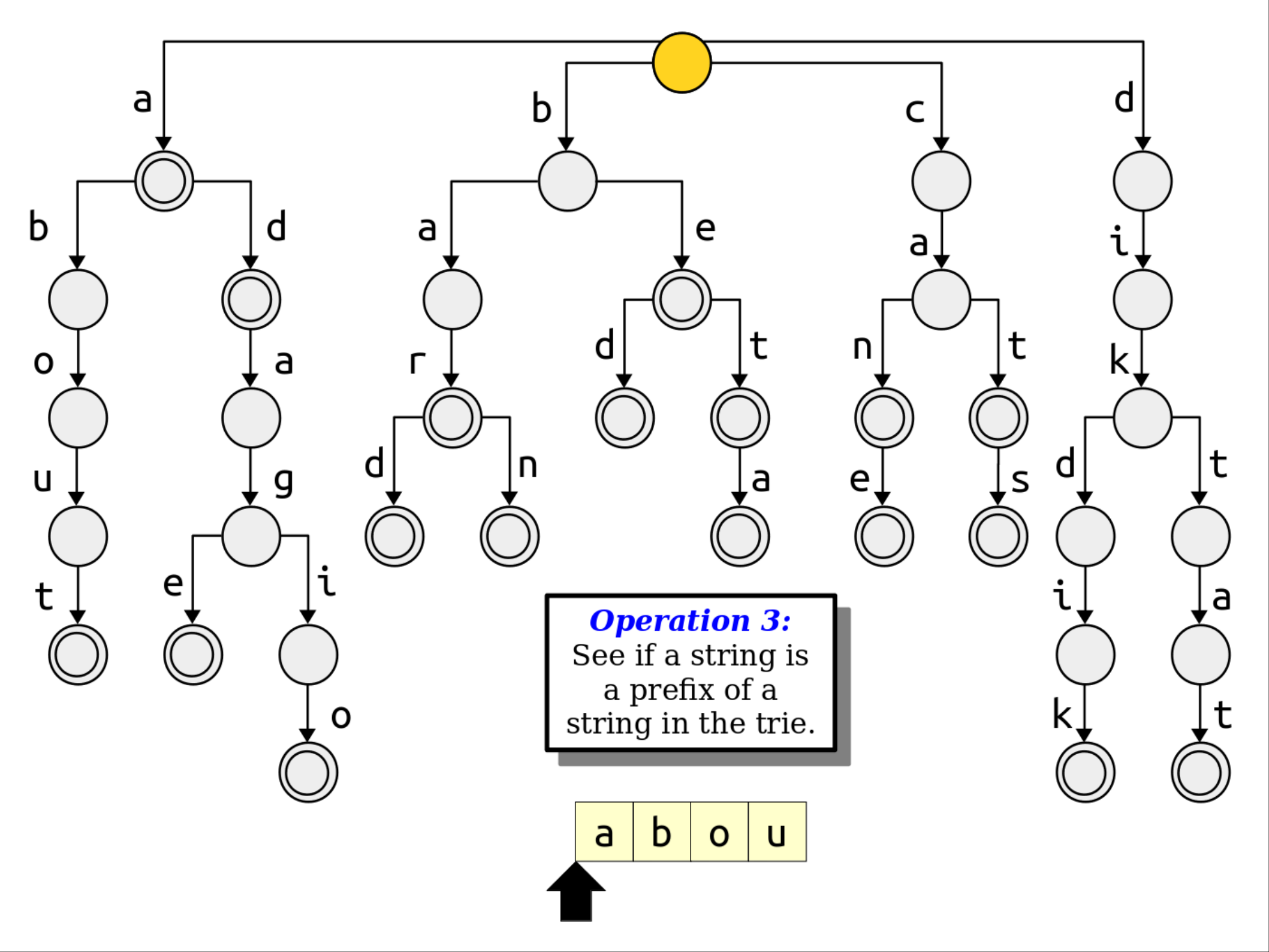
Operation 2:
Add a string into the collection of words.

b e



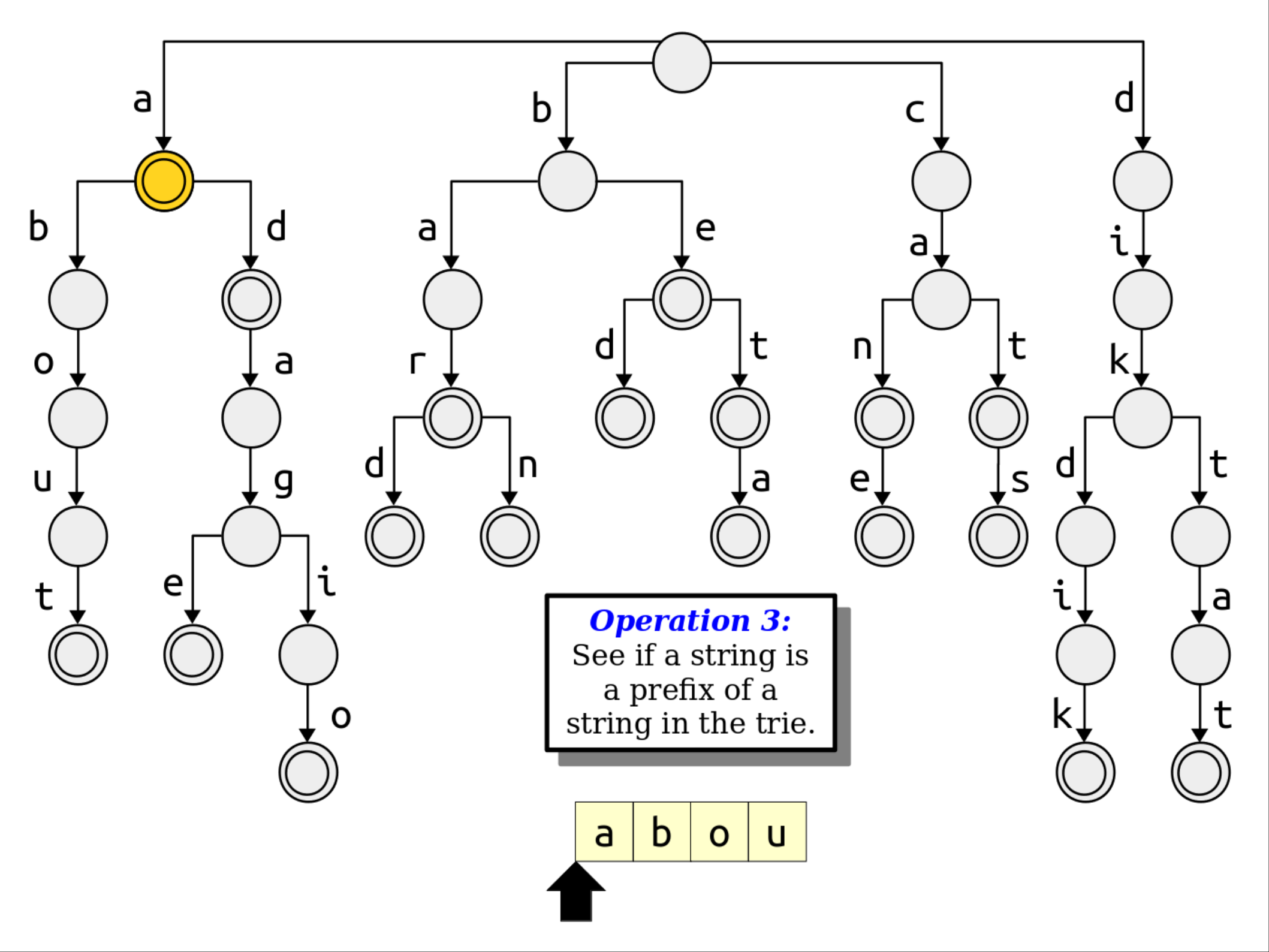
Operation 3:
See if a string is a prefix of a string in the trie.

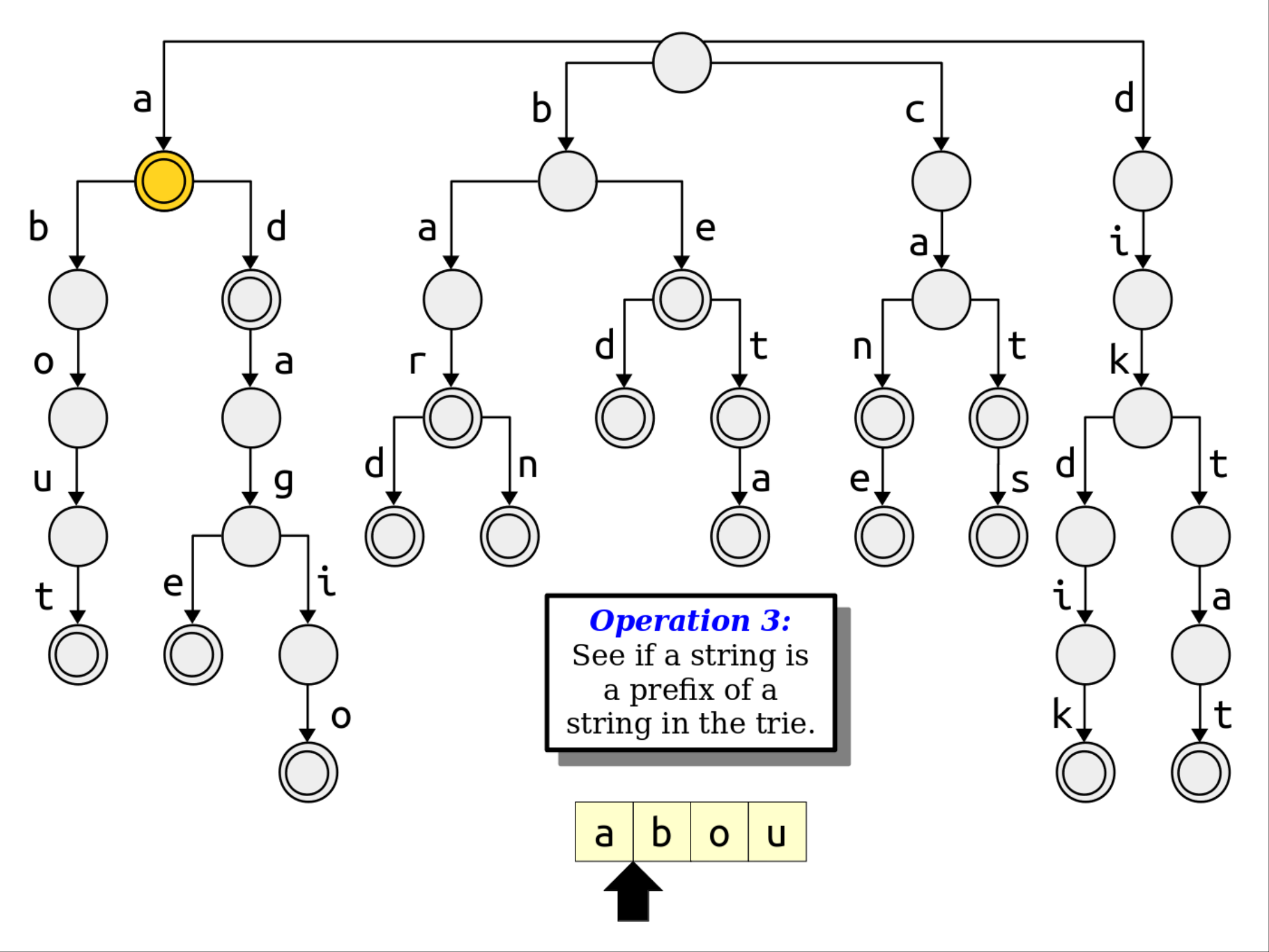

a
b
c
d
b
d
a
e
a
i
o
a
r
d
t
n
t
k
u
g
d
n
d
t
e
s
t
e
i
a
i
o
k
t
Operation 3:
See if a string is a prefix of a string in the trie.
a
b
o
u



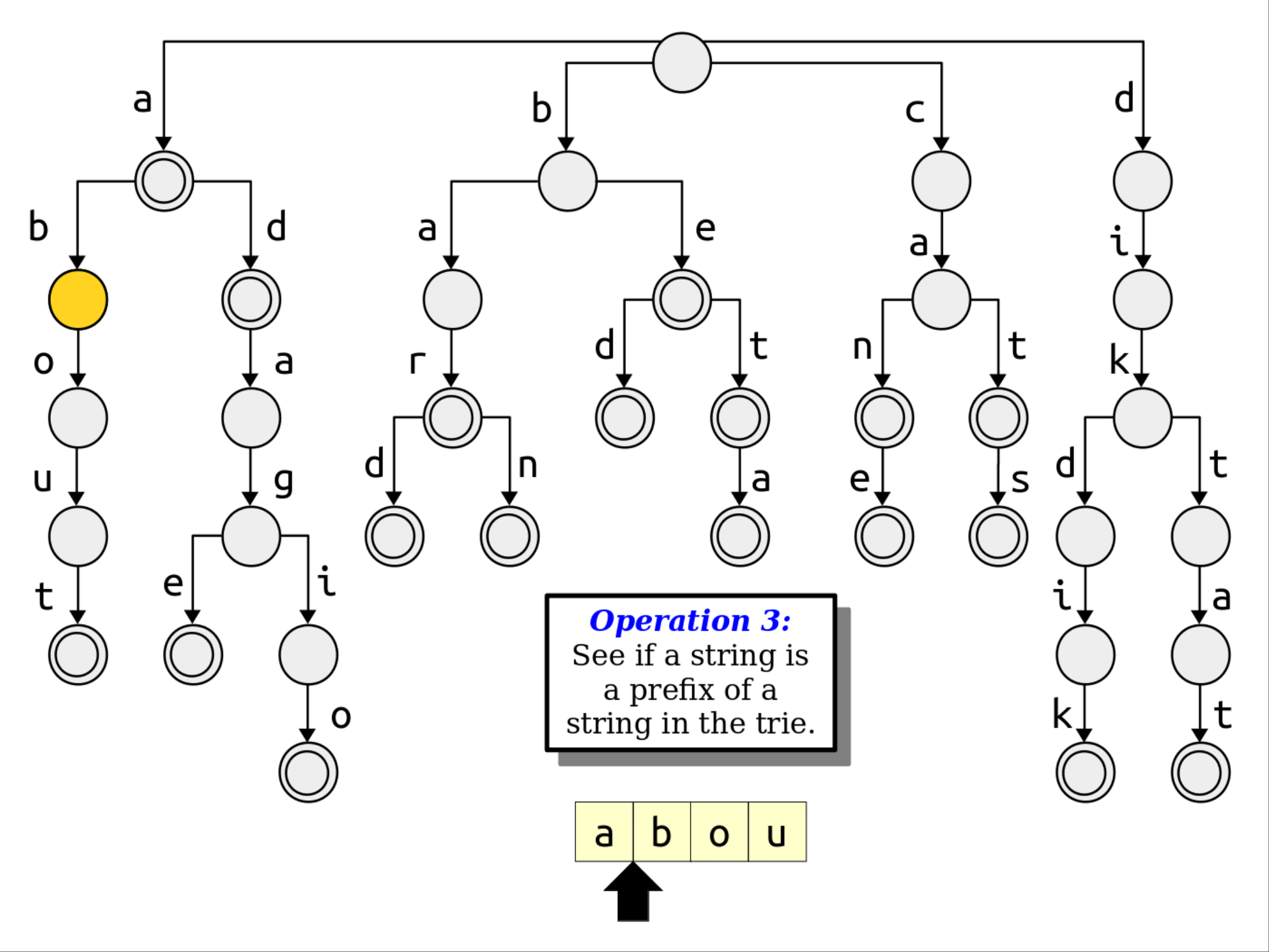
Operation 3:
See if a string is a prefix of a string in the trie.

a	b	o	u


a
b
c
d
b
d
a
e
a
i
o
a
r
d
t
n
t
k
u
g
d
n
d
t
e
s
t
e
i
i
a
o
k
t
Operation 3:
See if a string is a prefix of a string in the trie.
a
b
o
u

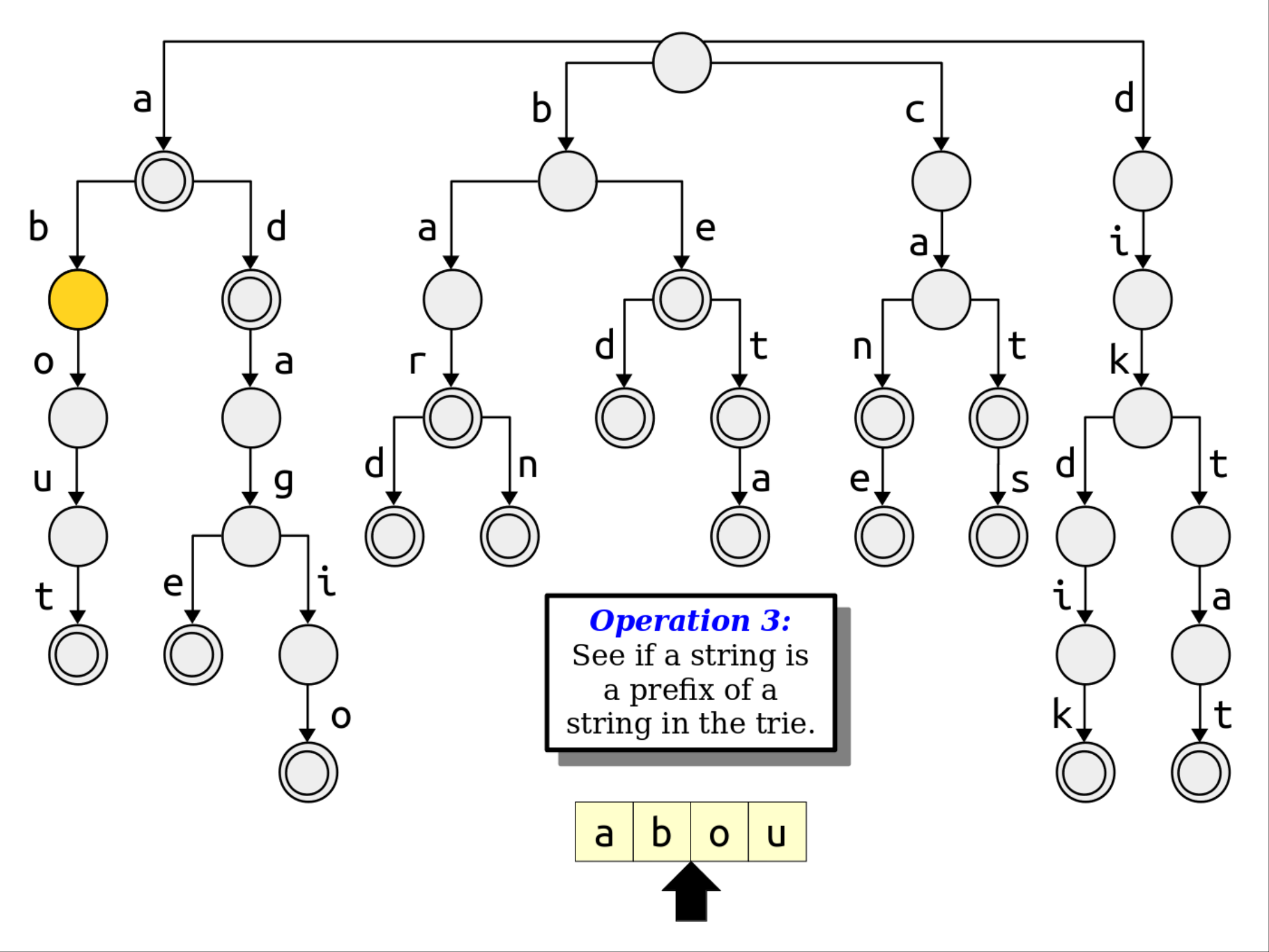


Operation 3:
See if a string is a prefix of a string in the trie.

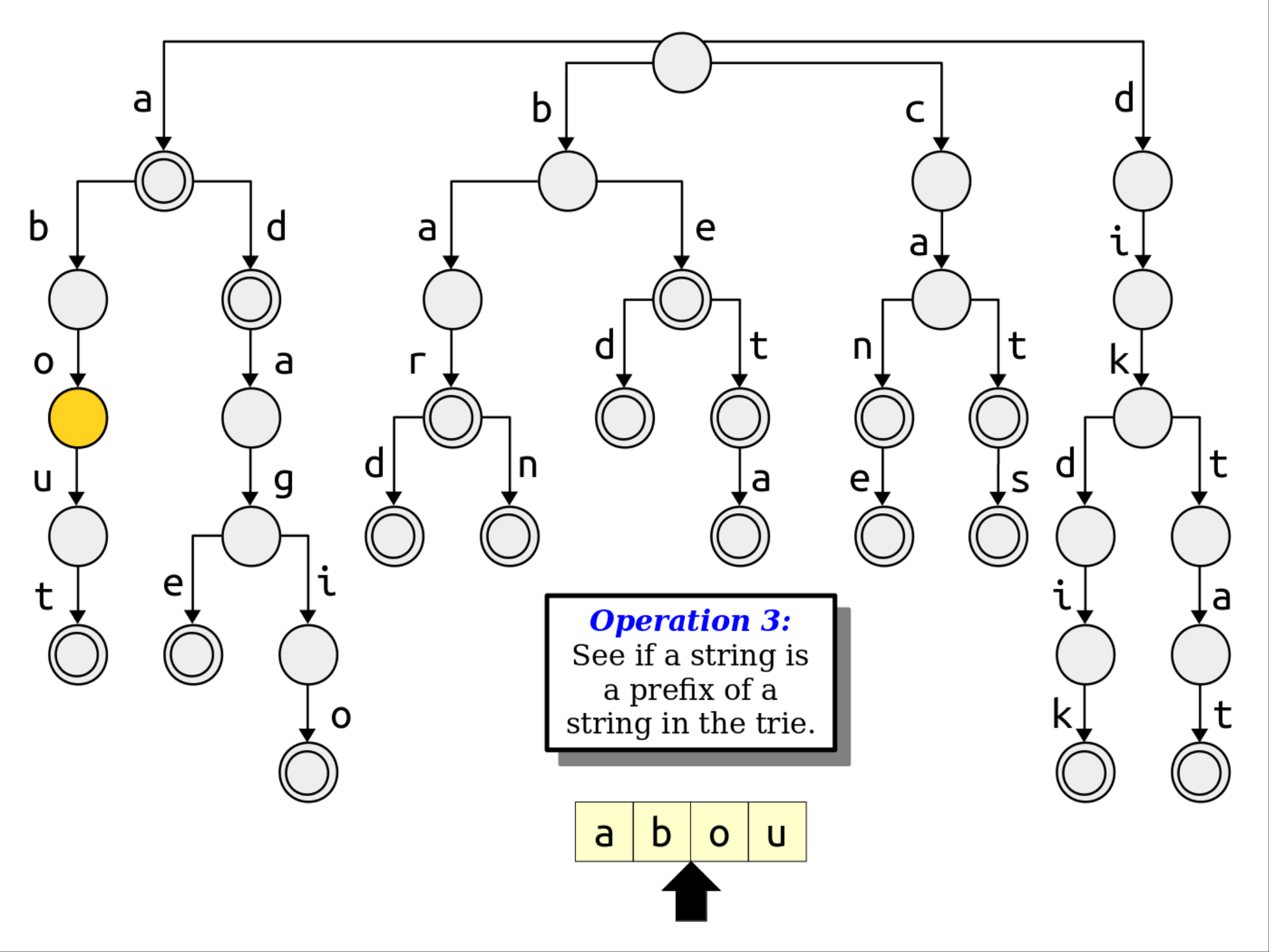


Operation 3:
See if a string is a prefix of a string in the trie.

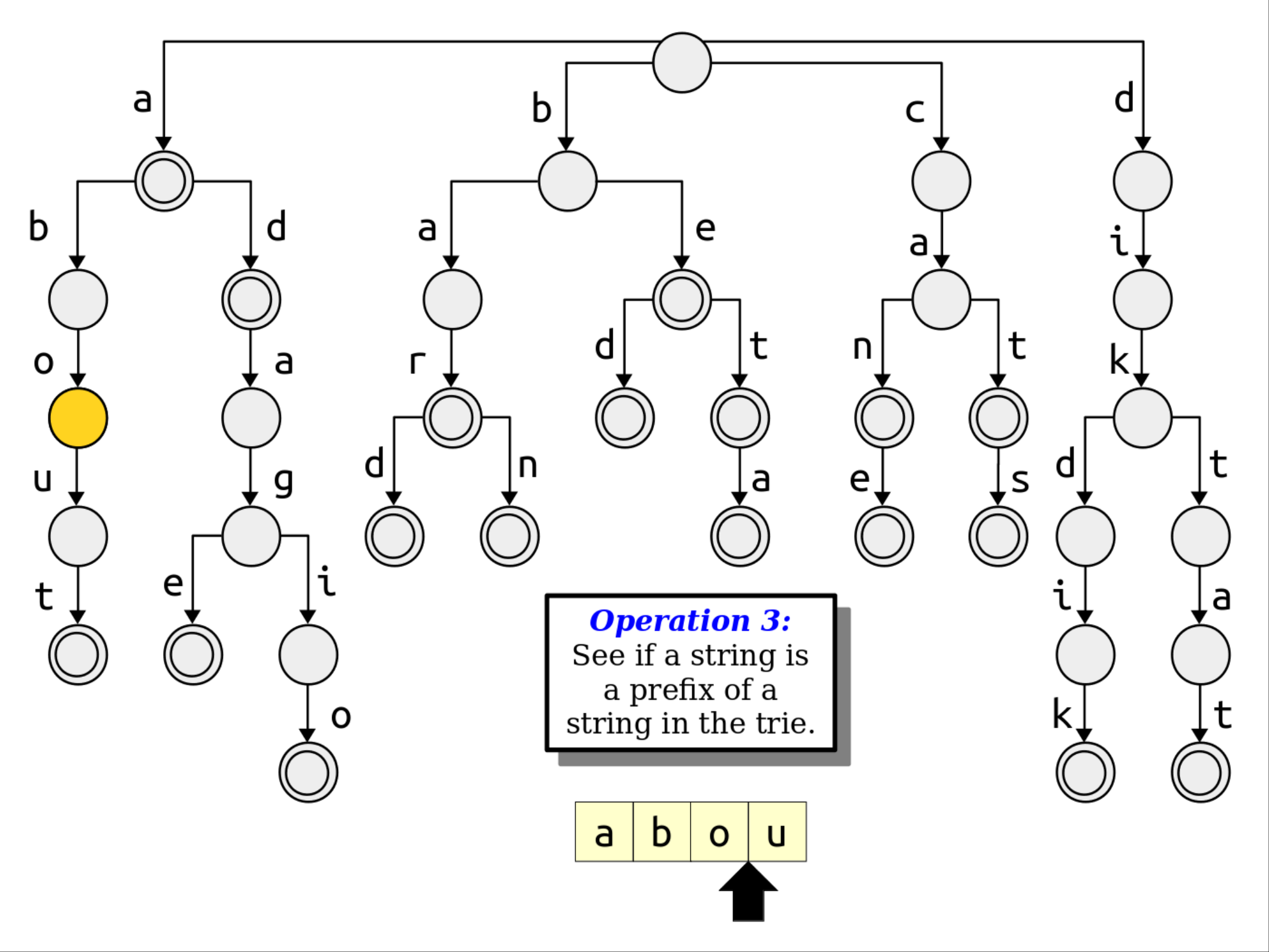
a	b	o	u



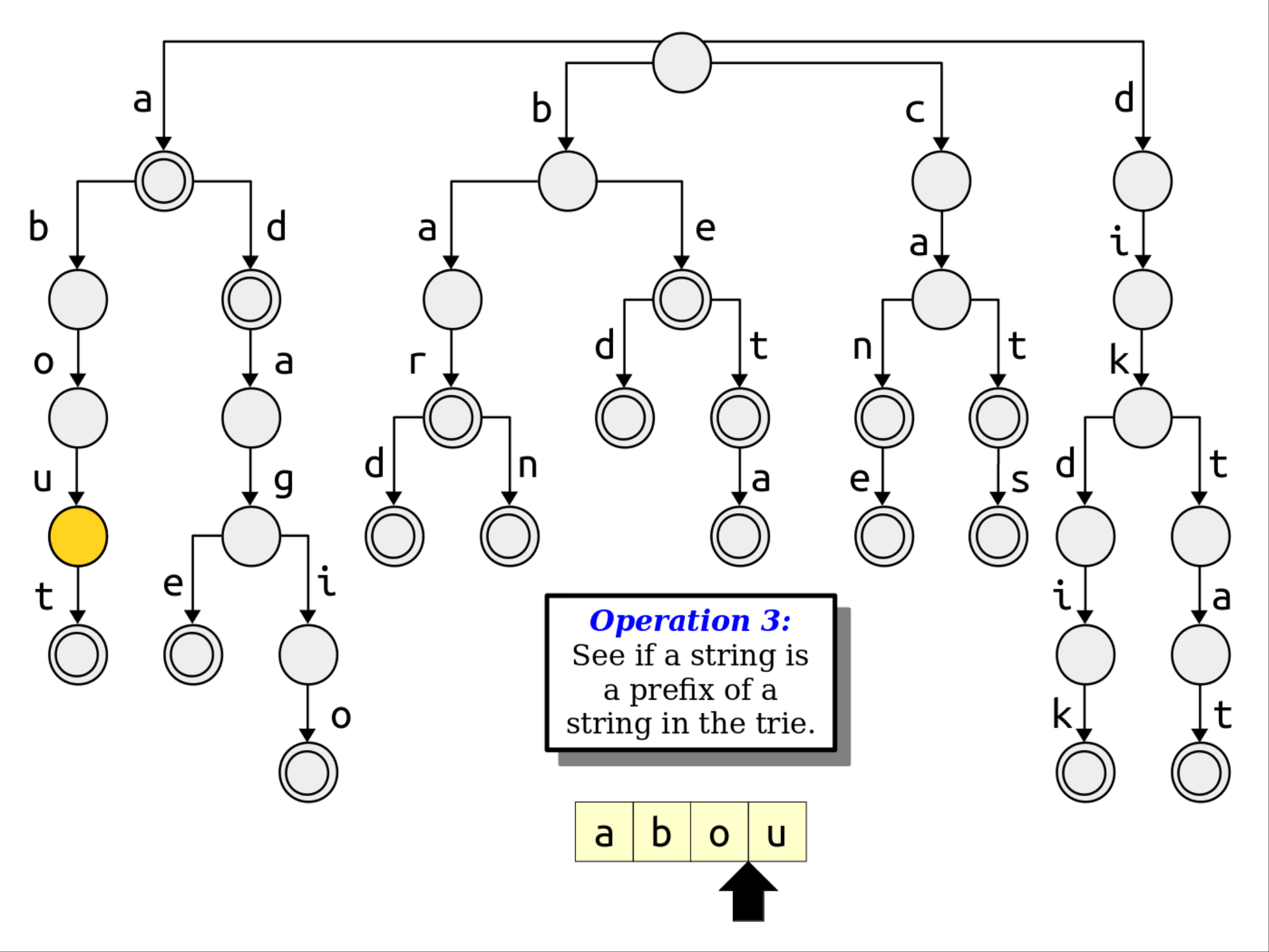
Operation 3:
See if a string is a prefix of a string in the trie.



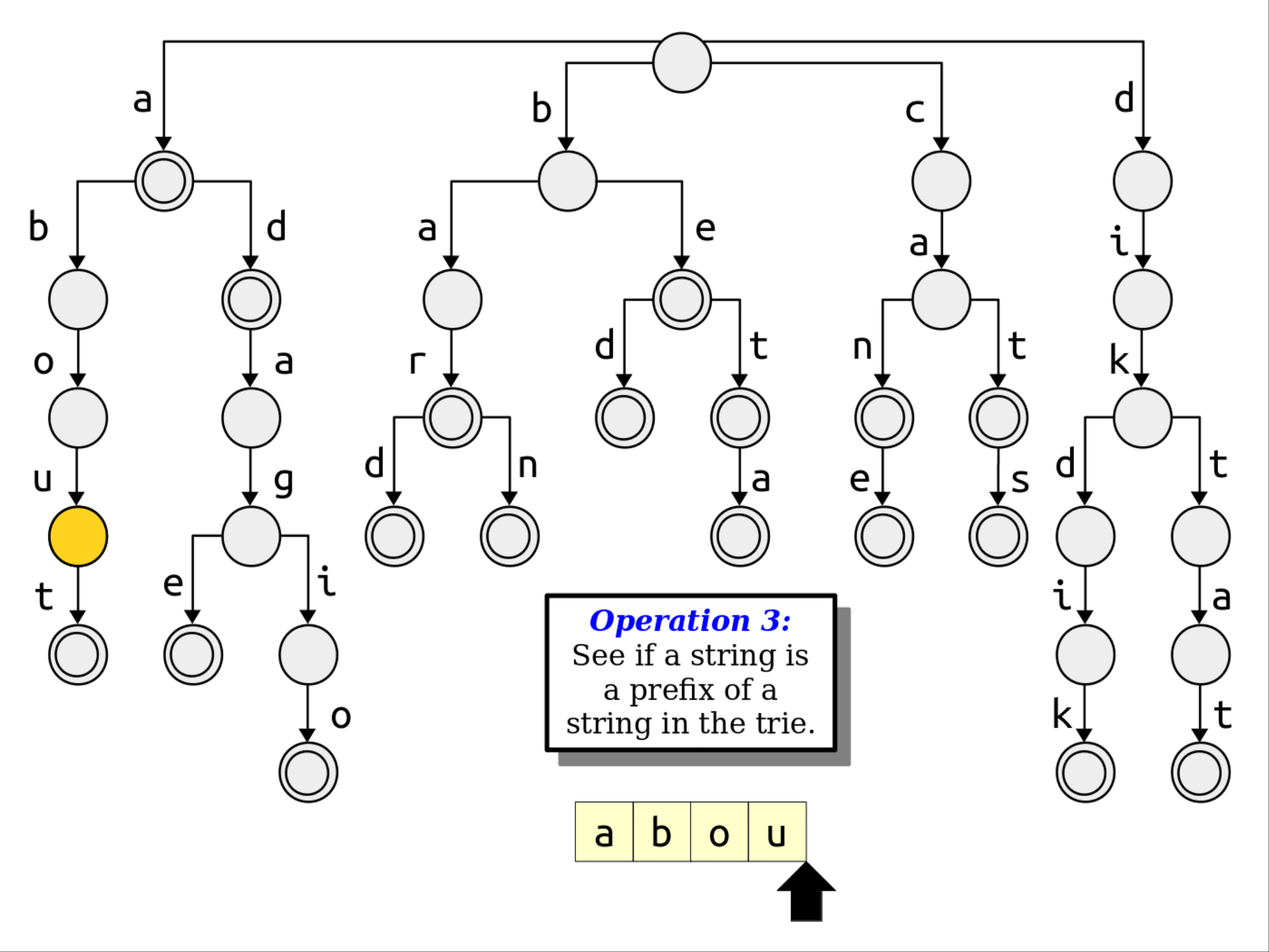
Operation 3:
See if a string is a prefix of a string in the trie.



Operation 3:
See if a string is a prefix of a string in the trie.

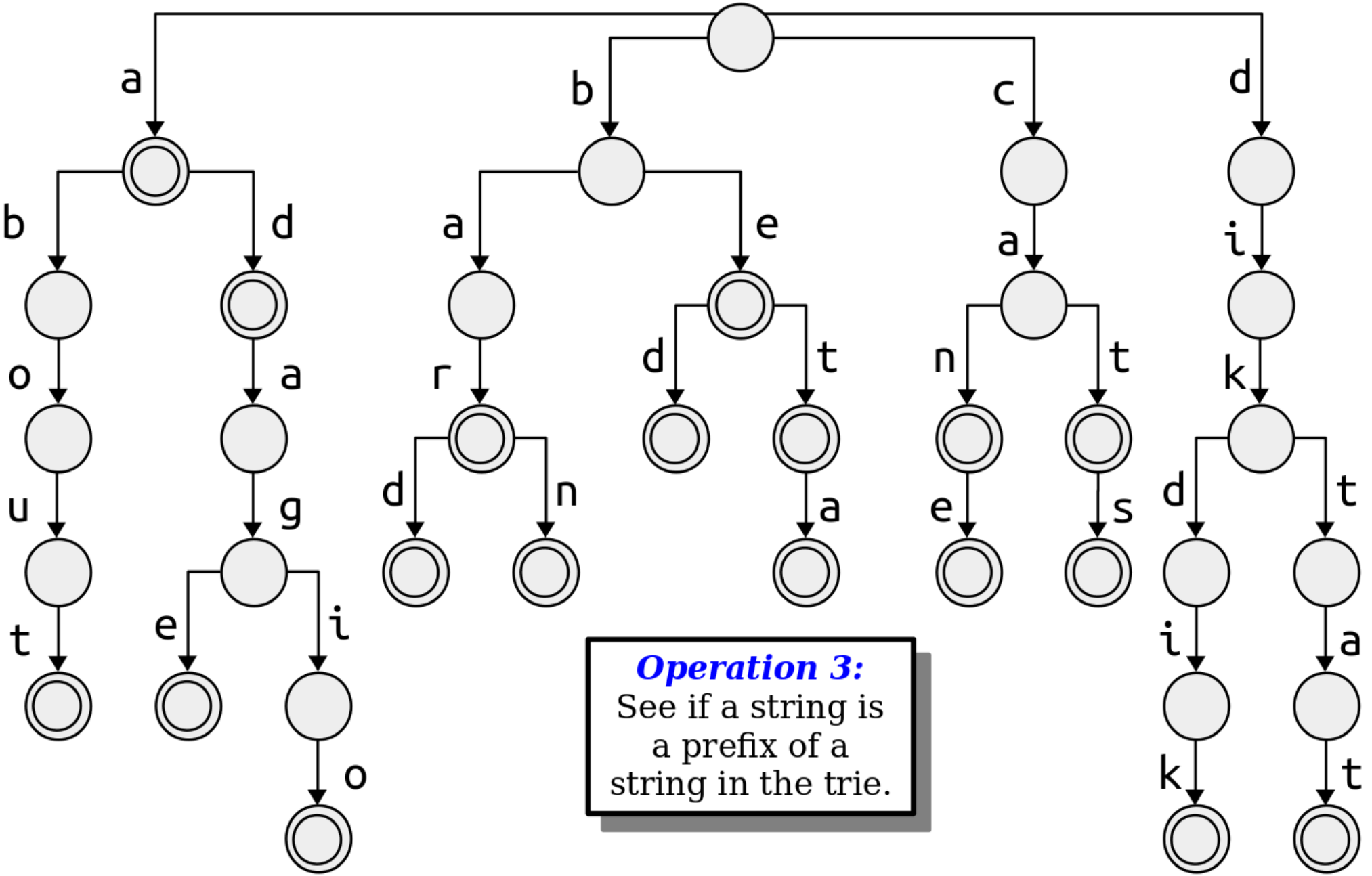


Operation 3:
See if a string is a prefix of a string in the trie.

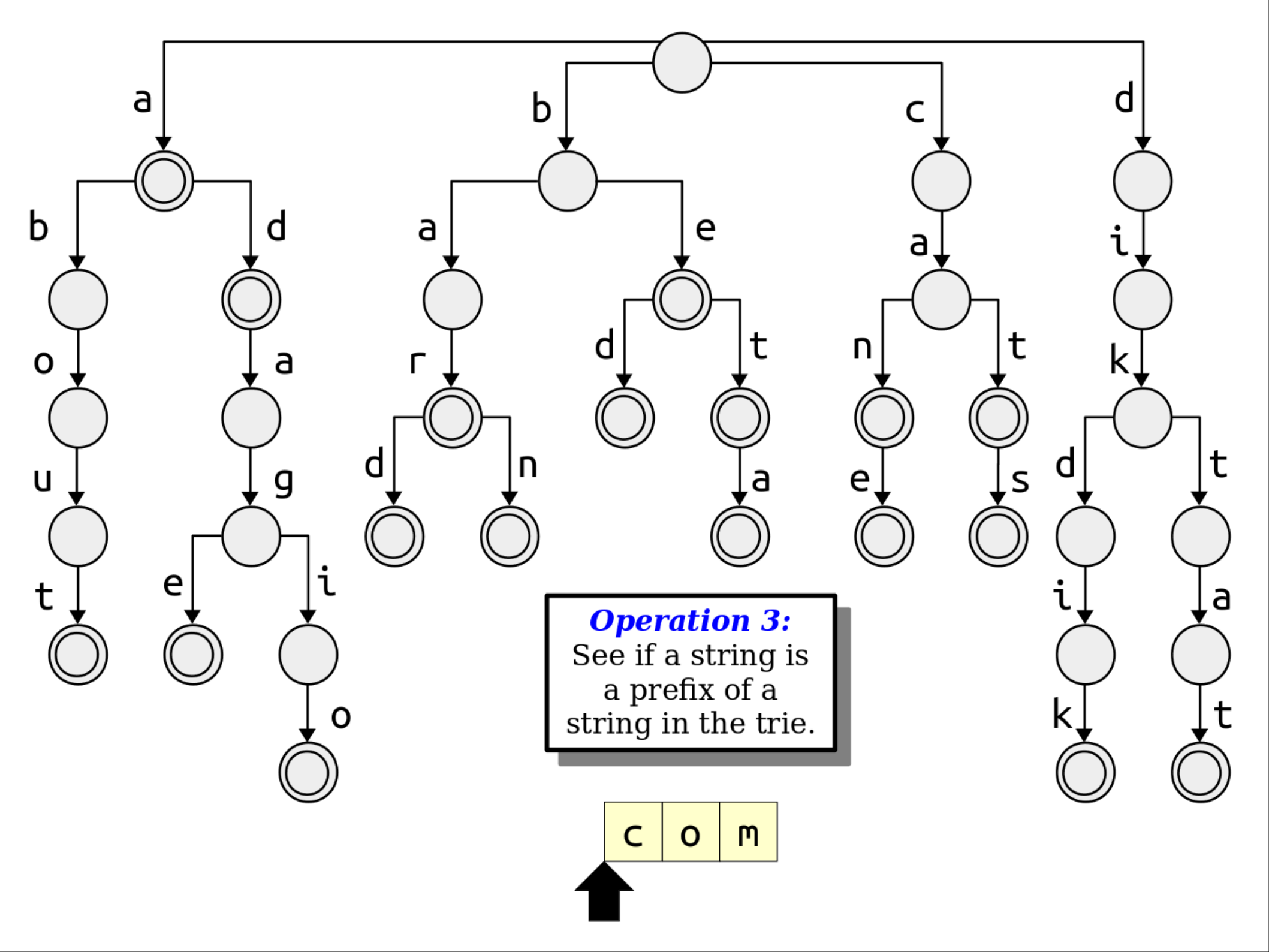


Operation 3:
See if a string is a prefix of a string in the trie.

a	b	o	u

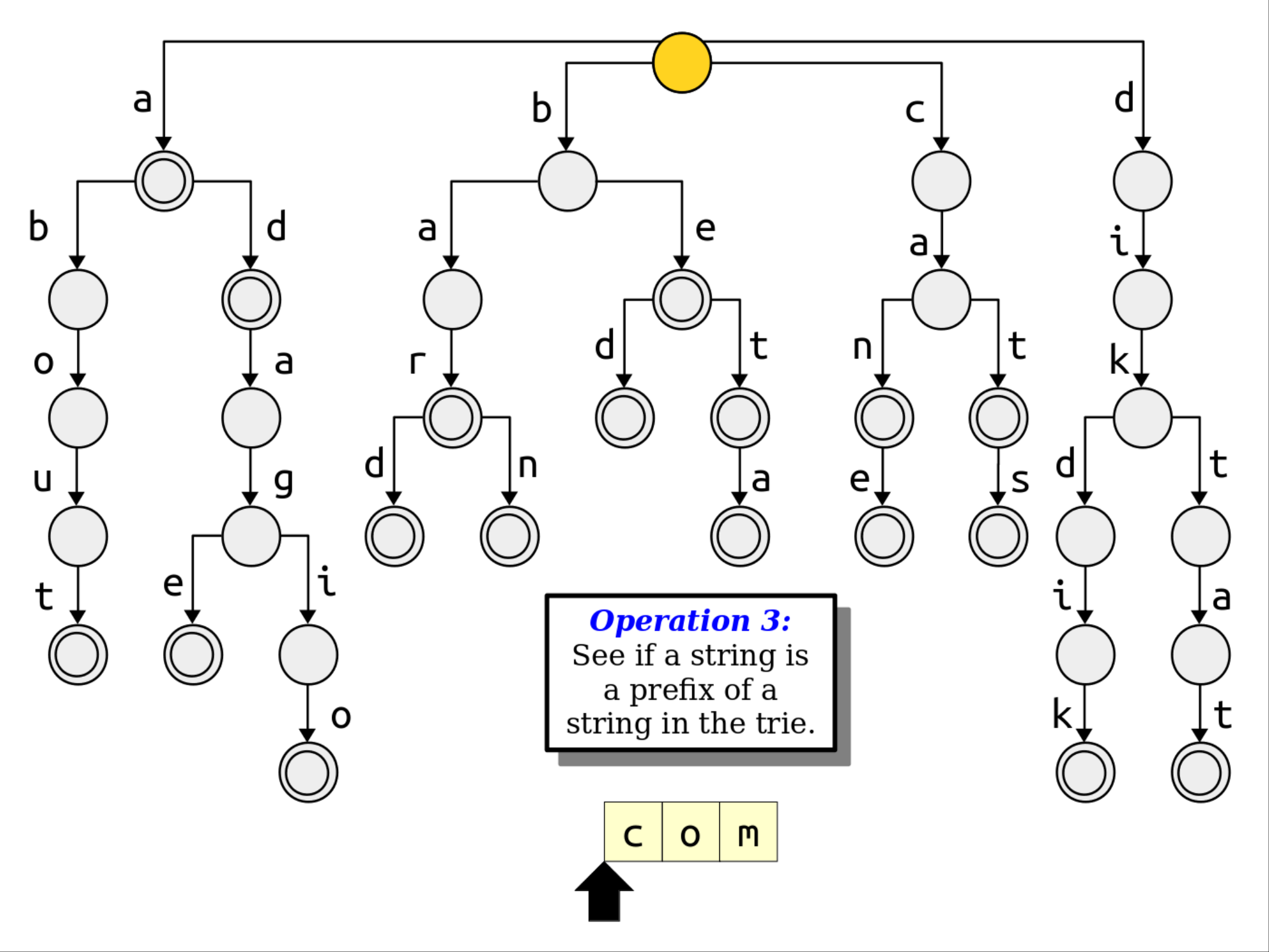


Operation 3:
See if a string is a prefix of a string in the trie.



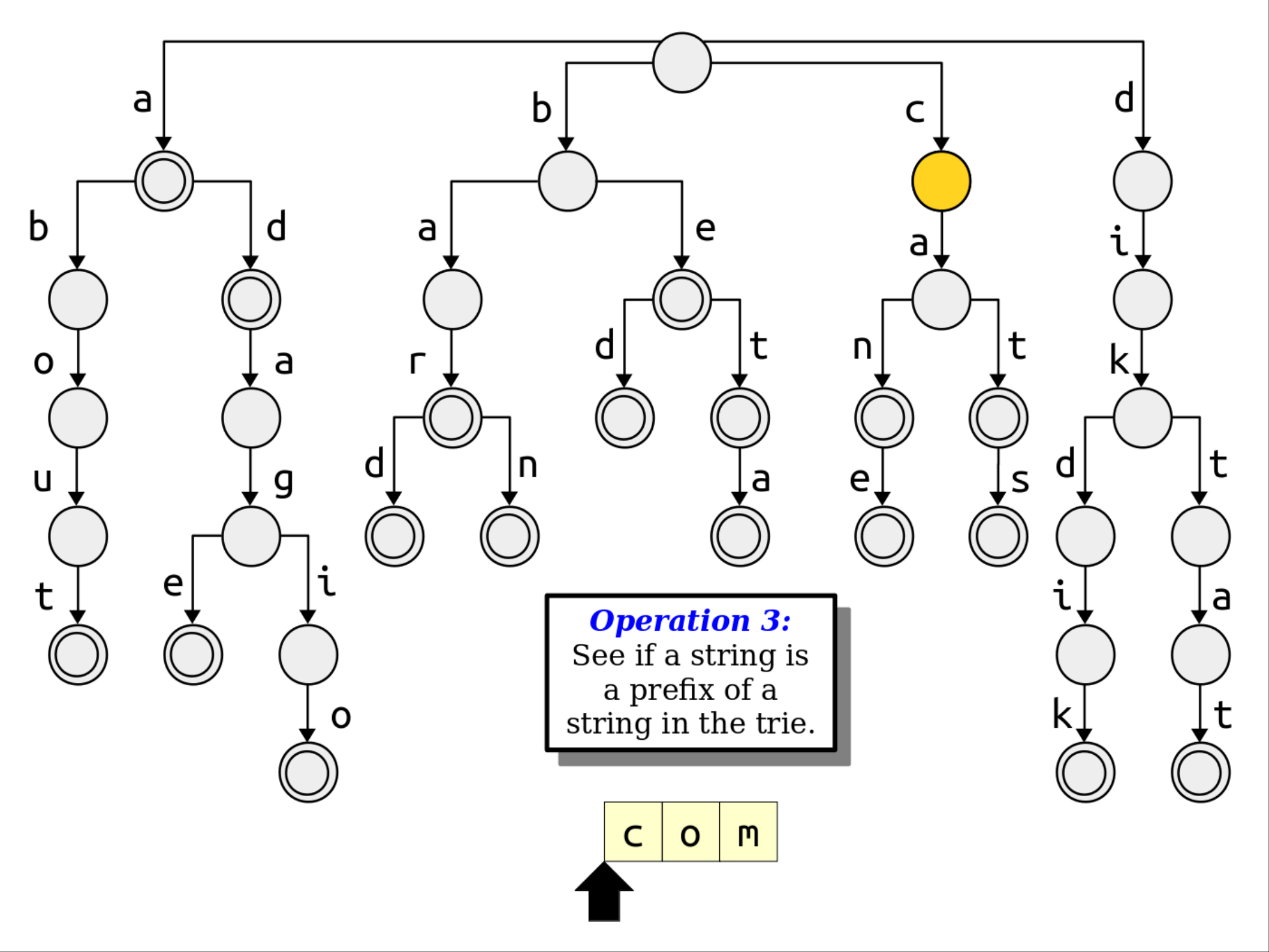
Operation 3:
See if a string is a prefix of a string in the trie.

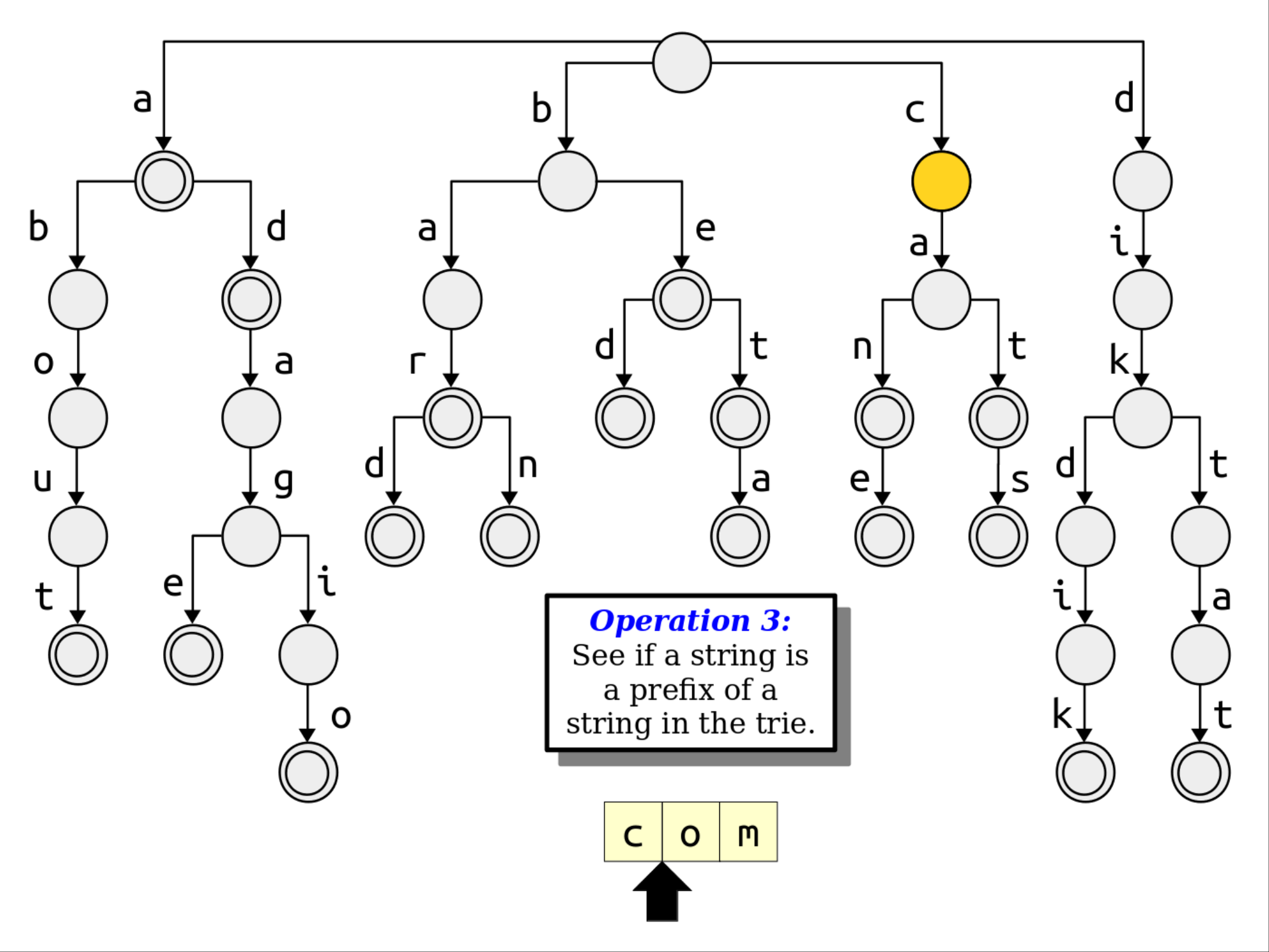
c	o	m



Operation 3:
See if a string is a prefix of a string in the trie.

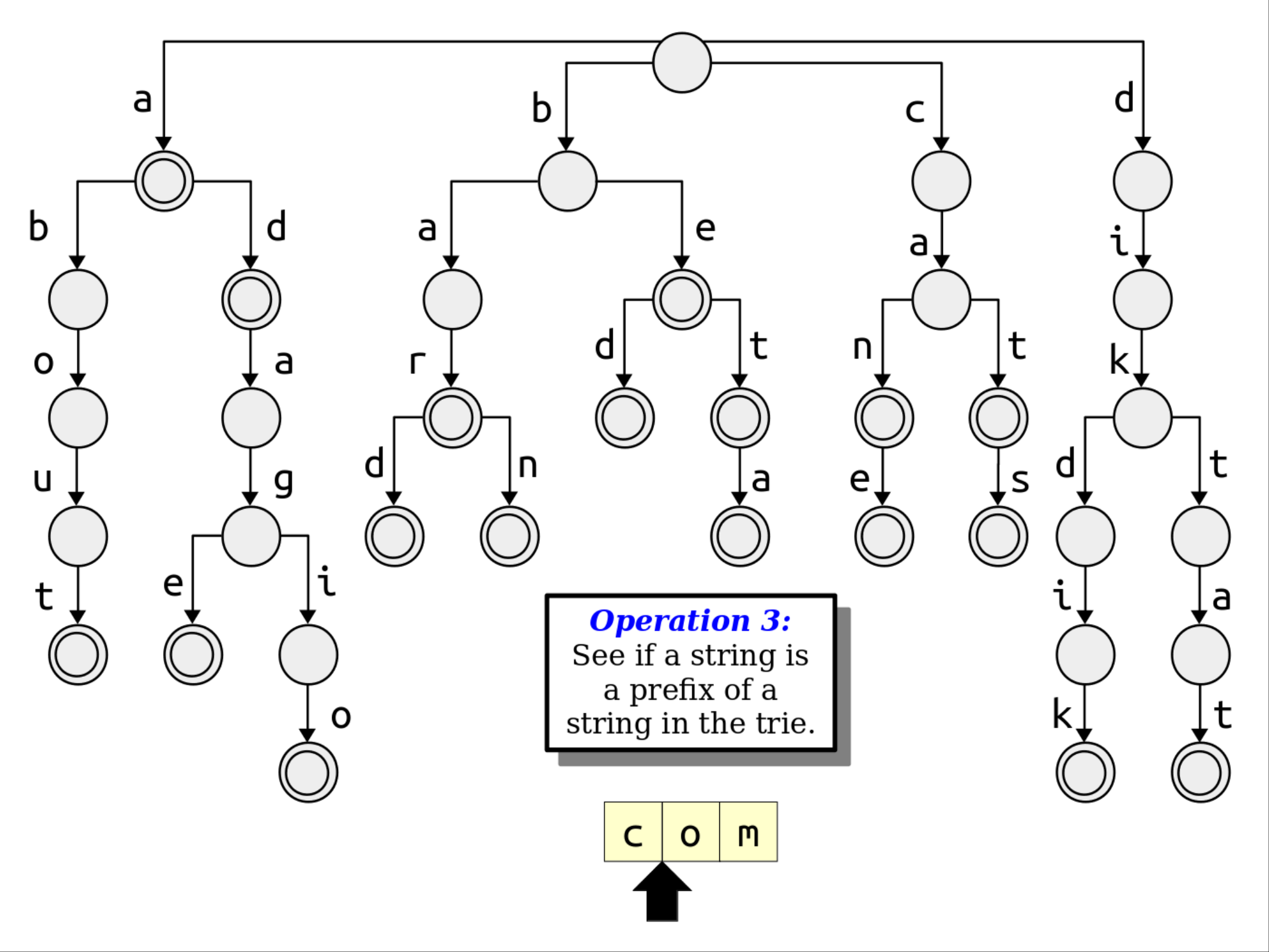
c	o	m


a
b
c
d
b
d
a
e
a
i
o
a
r
d
t
n
t
k
u
g
d
n
d
t
e
s
t
e
i
a
i
o
k
t
Operation 3:
See if a string is a prefix of a string in the trie.
c
o
m



Operation 3:
See if a string is a prefix of a string in the trie.

c	o	m



Operation 3:
See if a string is a prefix of a string in the trie.

c	o	m

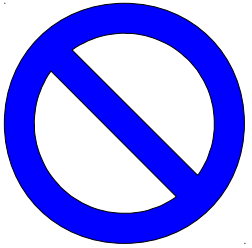
Other Operations on Tries

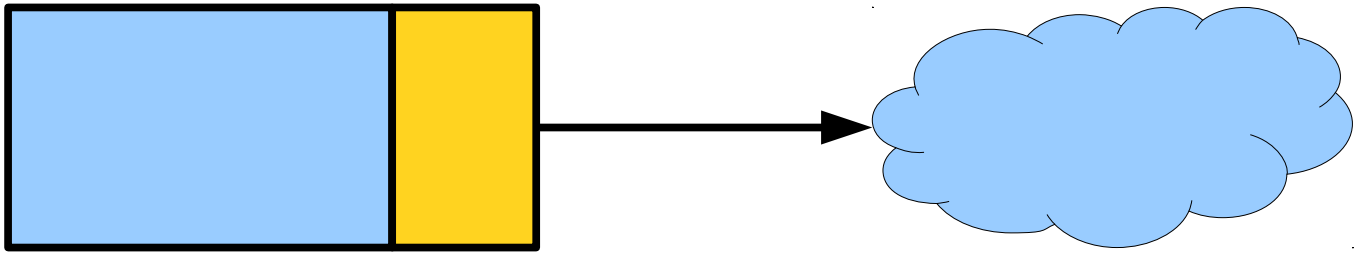
- Find all strings in the trie that start with a given prefix.
  - ***How might you implement this?***
- Print all strings in sorted order.
  - ***How might you implement this?***
- Find the first string that’s alphabetically before or after another.
  - ***How might you implement this?***

A Useful Perspective

A Linked List is Either…

…an empty list, represented by `nullptr`, or…



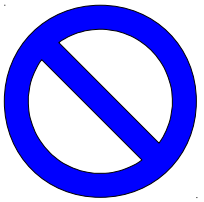


a single linked list cell that points…

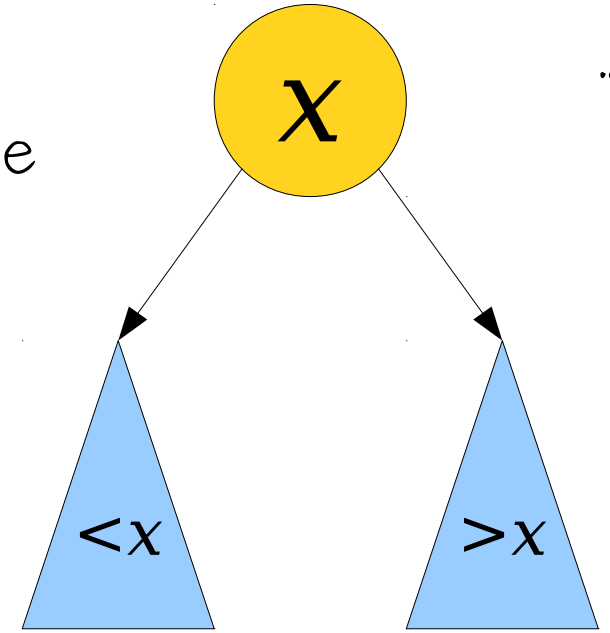
… at another linked list.

A Binary Search Tree Is Either…

an empty tree,
represented by
`nullptr`, or…



… a single node,
whose left subtree
is a BST of
smaller values …

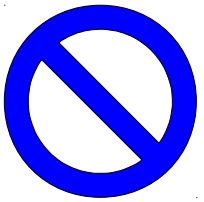


… and whose right
subtree is a BST
of larger values.

A Trie is Either…

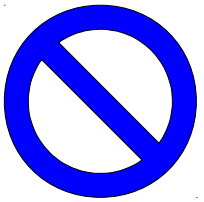
an empty trie,
represented by

`nullptr`

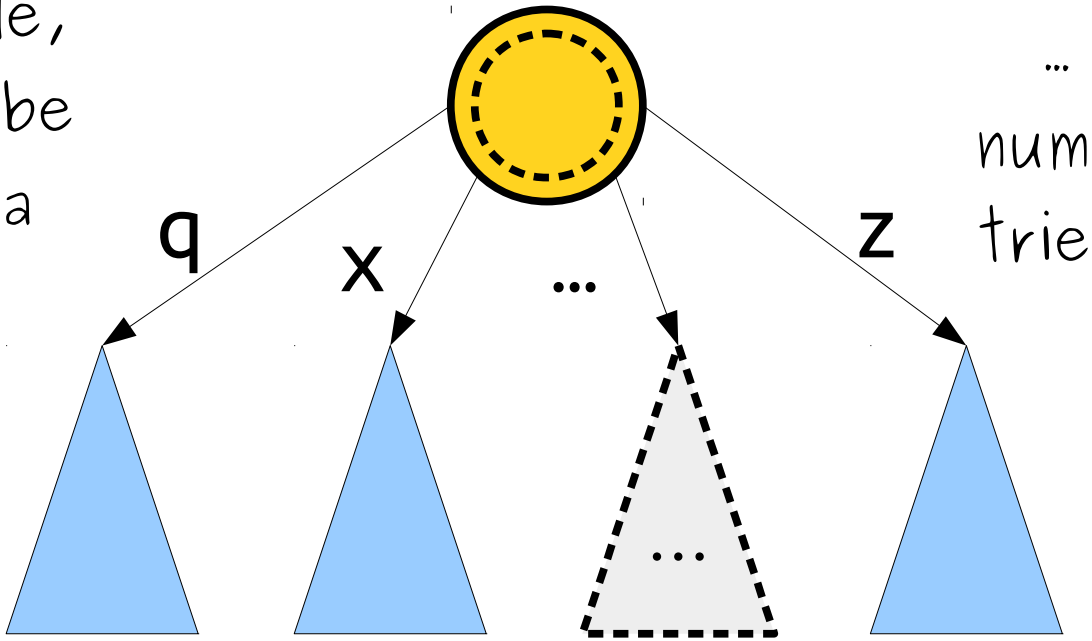


A Trie is Either…

an empty trie, represented by **`nullptr`**, or…



a single node, which might be marked as a word…



… with some number of child tries labeled by letters.

Assignment 6: Think through these design decisions!

```
struct Cell {
    Type value;
    Cell* next;
};
```

Singly-Linked List

```
struct Node {
    Type value;
    Node* left;
    Node* right;
};
```

Binary Search Tree

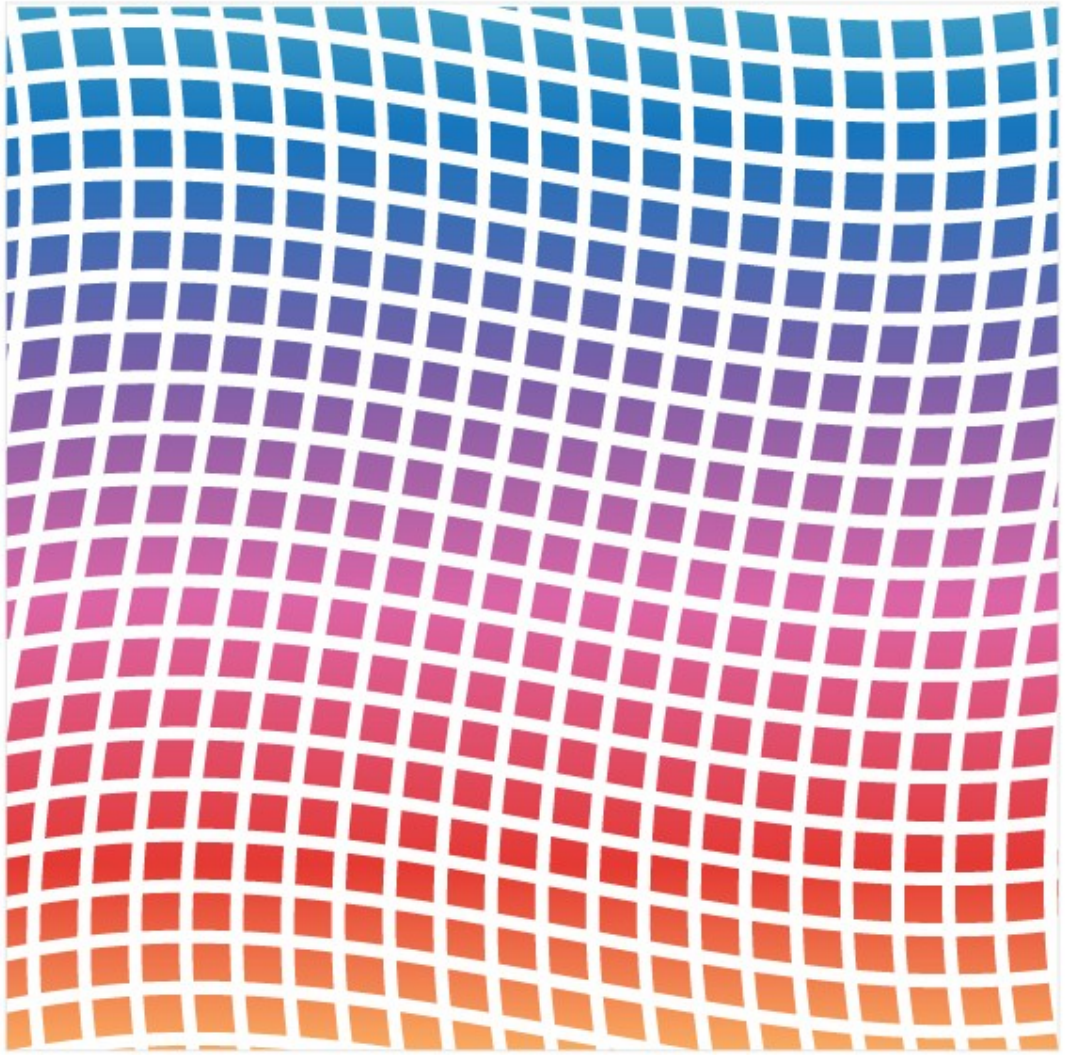
```
struct Name? {

    /* ? */

};
```

Trie

Time-Out for Announcements!



oSTEM

CAREER

OFFICE HOURS

Friday, 3/1 4pm-5pm
QSpot (2nd floor Fire Truck House)
Snacks, Headshots, & Advice!

Assignment 6

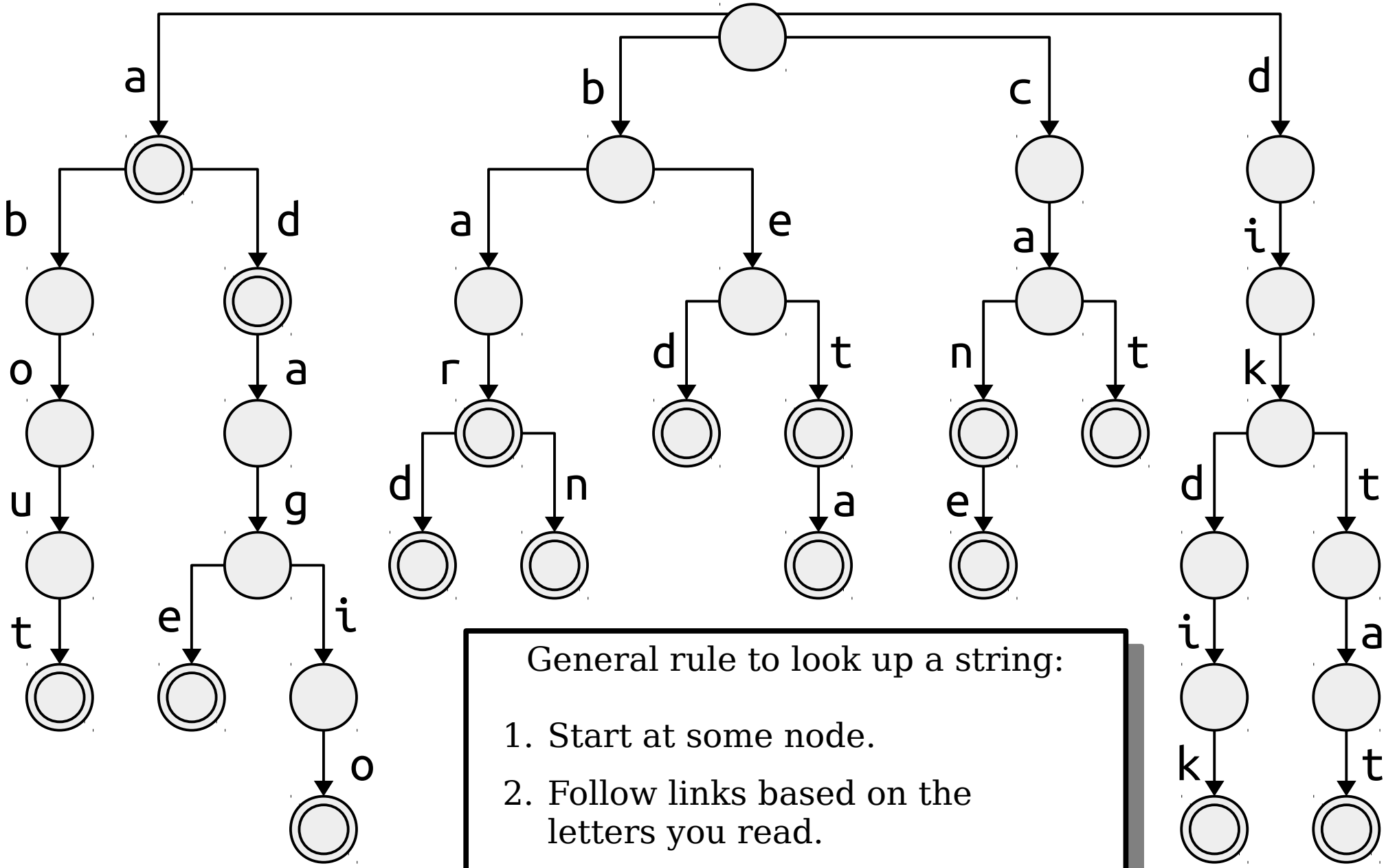
- Assignment 6 (***MiniBrowser***) goes out today. It's due one week from Friday at the start of class.
  - Play around with linked lists and tree data structures!
  - Build integral pieces of a larger system!
  - See why all this stuff matters.
- YEAH hours will be held today at 5:00PM in 380-380Y. Slides will be posted.

Back to CS106B!

Twists on Tries

(a sneak peek of beautiful CS concepts!)

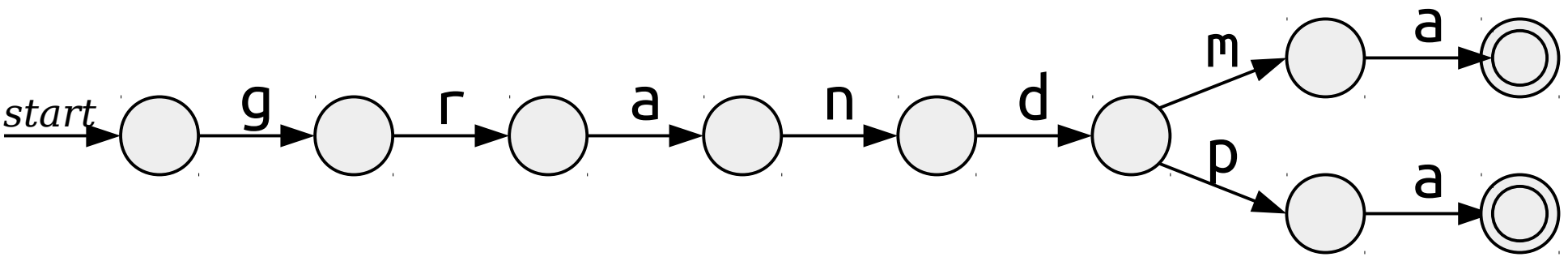
Twist: ***Finite Automata***



General rule to look up a string:

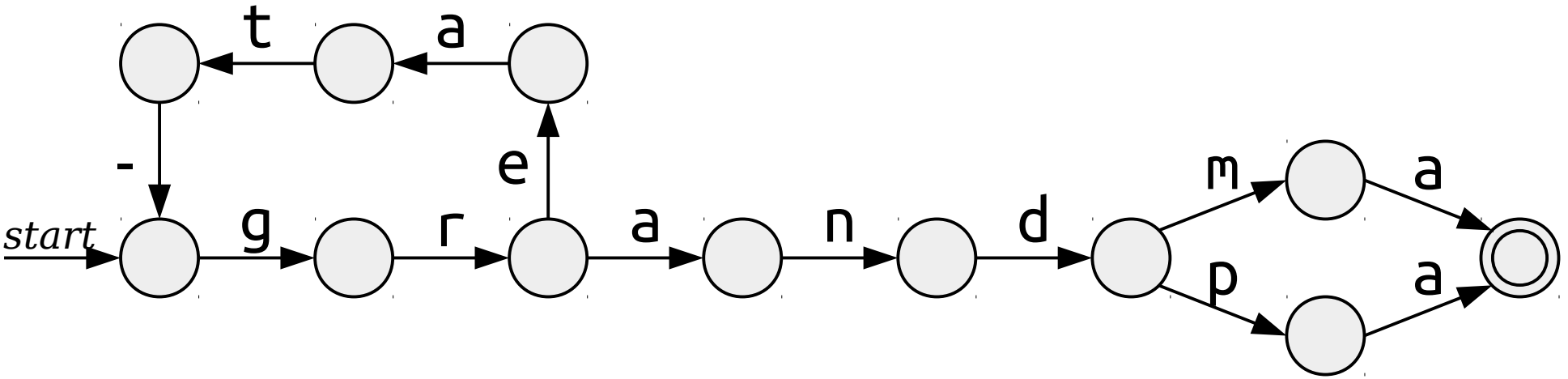
1. Start at some node.
2. Follow links based on the letters you read.
3. The string is there if you don’t get stuck and land at a double circle.

Breaking the Rules




```
grandma
grandpa
```

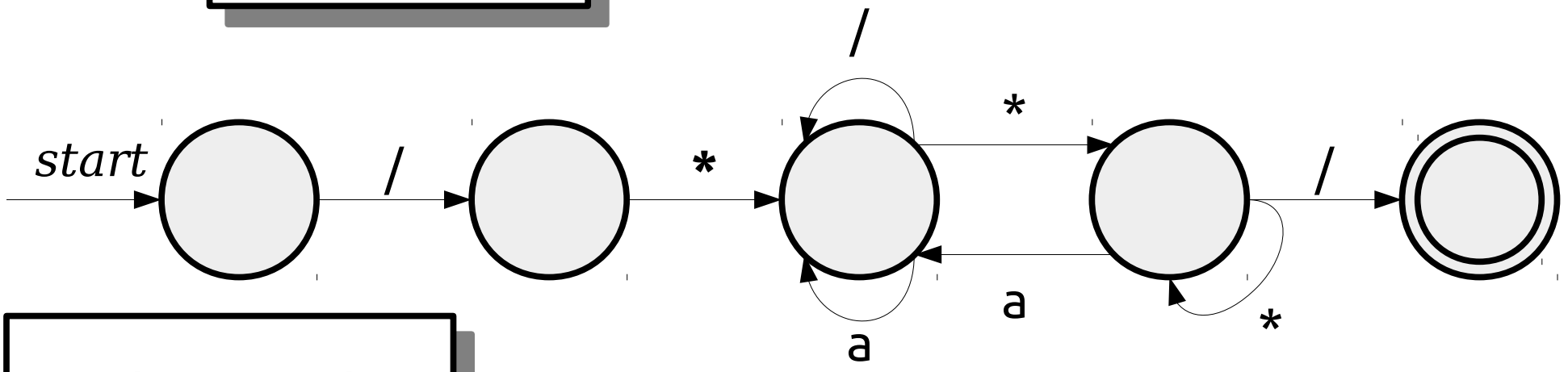
Breaking the Rules




```
grandma
grandpa
great-grandma
great-great-grandpa
great-great-great-great-grandpa
…
```

This isn't a tree, but we can still follow the same rules!

What sorts of strings does this weird thingy contain?



/*aaaa*/
/***aaaa***/
/*aaa*aaa*/
/*********/
...

Finite Automata

- A ***finite automaton*** is a generalization of a trie.
- It's not necessarily a tree; there can be circular paths, places where branches come together, etc.
- Finite automata power many compilers and pattern-matching tools.
- Want to learn more? ***Take CS103!***

Twist: ***Suffix Trees***

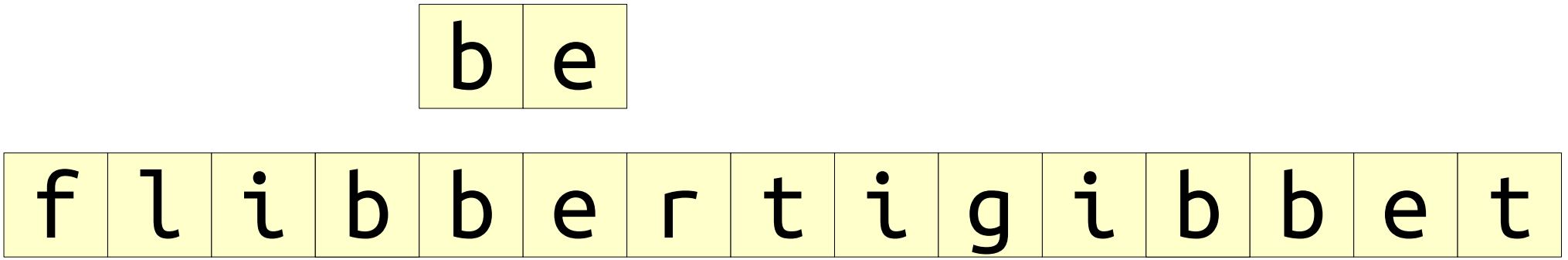
Cancer cells often have multiple repeated copies the same gene.

Given a cancer genome (length ~3,000,000,000 letters)
and a gene, count the occurrences of that gene.

A Fundamental Theorem

- The ***fundamental theorem of stringology*** says that, given two strings *w* and *x*, that

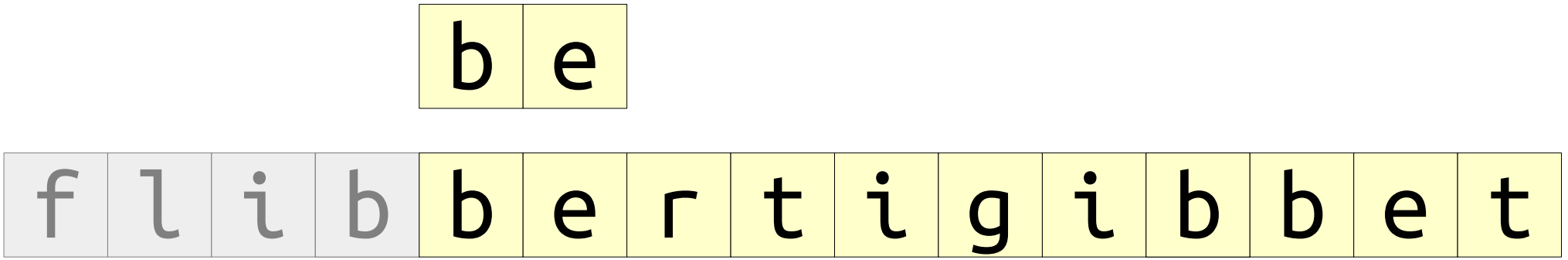
_w_ is a substring of _x_
if and only if
_w_ is a prefix of a suffix of _x_



A Fundamental Theorem

- The ***fundamental theorem of stringology*** says that, given two strings *w* and *x*, that

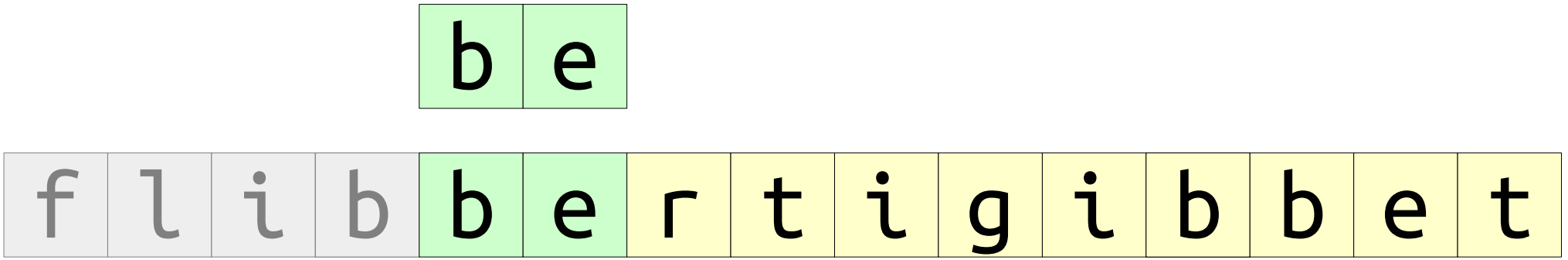
_w_ is a substring of _x_
if and only if
_w_ is a prefix of a suffix of _x_



A Fundamental Theorem

- The ***fundamental theorem of stringology*** says that, given two strings *w* and *x*, that

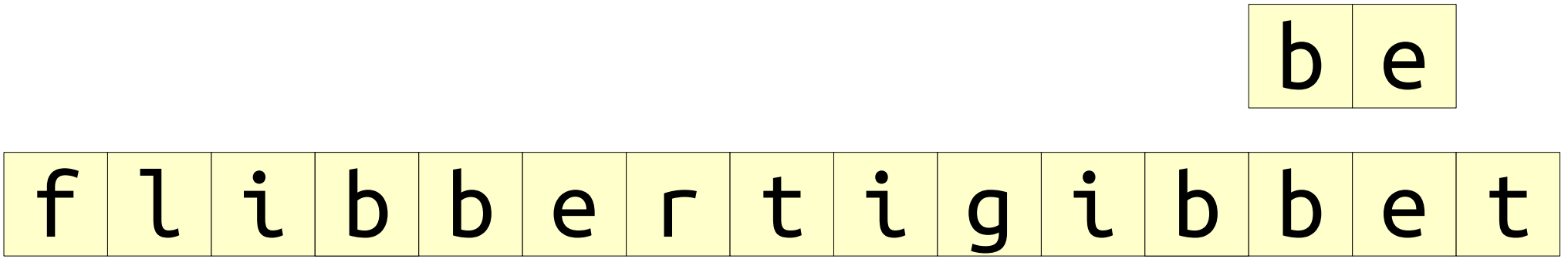
_w_ is a substring of _x_
if and only if
_w_ is a prefix of a suffix of _x_



A Fundamental Theorem

- The ***fundamental theorem of stringology*** says that, given two strings *w* and *x*, that

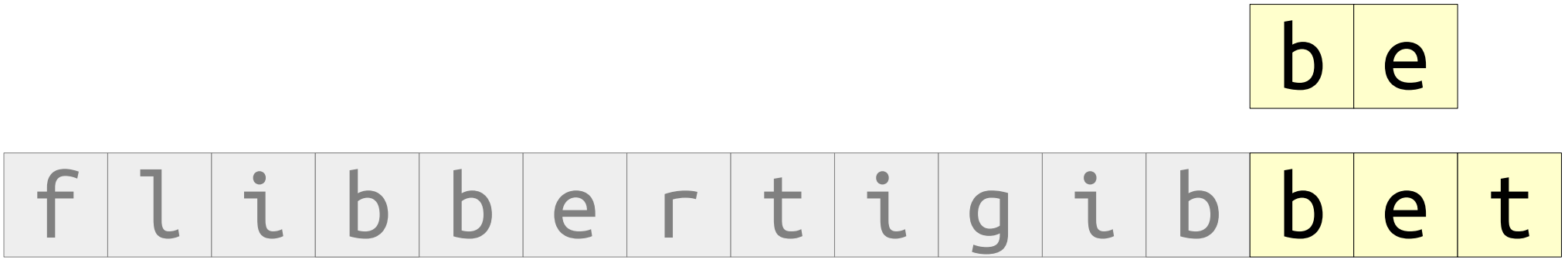
_w_ is a substring of _x_
if and only if
_w_ is a prefix of a suffix of _x_



A Fundamental Theorem

- The ***fundamental theorem of stringology*** says that, given two strings *w* and *x*, that

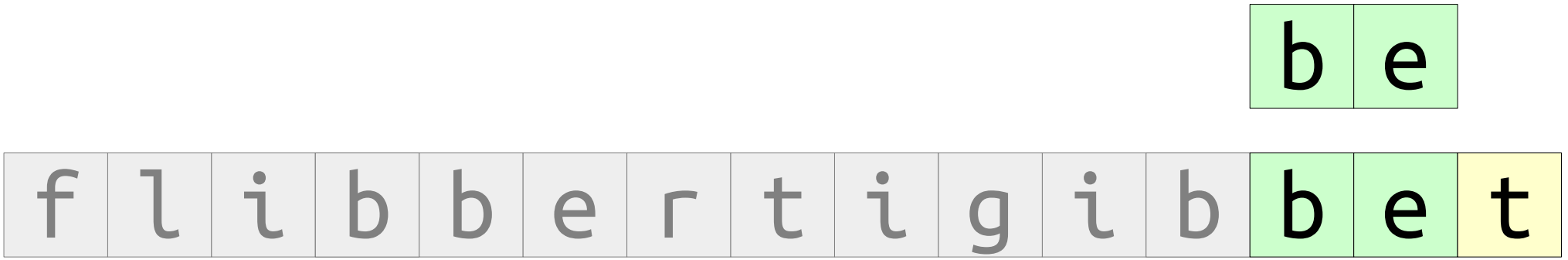
_w_ is a substring of _x_
if and only if
_w_ is a prefix of a suffix of _x_



A Fundamental Theorem

- The ***fundamental theorem of stringology*** says that, given two strings *w* and *x*, that

_w_ is a substring of _x_
if and only if
_w_ is a prefix of a suffix of _x_

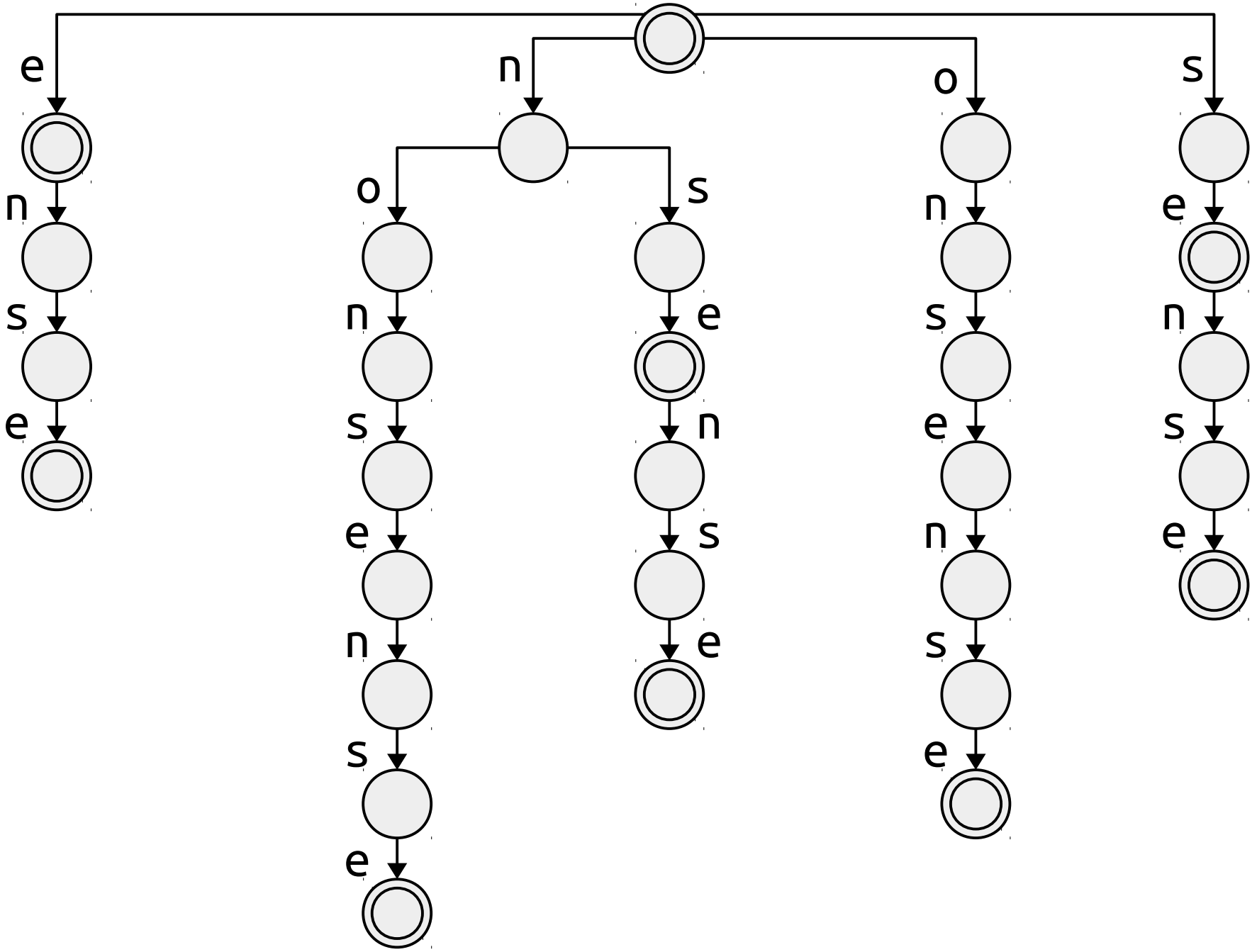


A Fundamental Theorem

- The ***fundamental theorem of stringology*** says that, given two strings *w* and *x*, that

  _w_ is a substring of _x_
  if and only if
  _w_ is a prefix of a suffix of _x_

- ***Recall:*** Tries make it really easy to check if something is a prefix of any number of strings.
- ***Idea:*** Store all the suffixes of a string in a trie!



nonsense

Suffix Trees

- With a lot of creativity, it's possible to compress the trie shown earlier to have only O(n) nodes.
- This is called a ***suffix tree*** and is a workhorse of a data structure.
- Want to learn more? Take CS166!

Next Time

- ***The Magic of Hash Functions***
  - A beautiful mathematical idea with incredible power.
- ***Hash Tables***
  - Surpassing BST performance!