Succinct Data Structures

Auto-completion as our target application

Rossano Venturini

rossano@di.unipi.it

autotrader

+Rossano Share

autotrader
autozone
auto loan calculator
autodesk

Learn more

Cookies help us deliver our services. By using our services, you agree to our use of cookies.
OK Learn more

New Cars, Used Cars - Find Cars at AutoTrader.com
www.autotrader.com/
Find used cars and new cars for sale at **AutoTrader**.com. With millions of cars, finding your next new car or used car and the car reviews and information you're ...
Used Car Research - Find Cars for Sale - Certified Pre-Owned Car - Sell a Car

Auto Trader UK – New & used cars for sale
www.autotrader.co.uk/
The UK's #1 site to buy and sell new and used cars, bikes, vans, trucks and caravans with over 350000 vehicles online. Check Car news, reviews and obtain ...
Used cars - Vans - Bikes - Used cars UK

Used cars - Find a used car for sale on Auto Trader
www.autotrader.co.uk/used-cars
Used cars for sale on **Auto Trader**, find the right used car for you at the UK's No.1 destination for motorists.

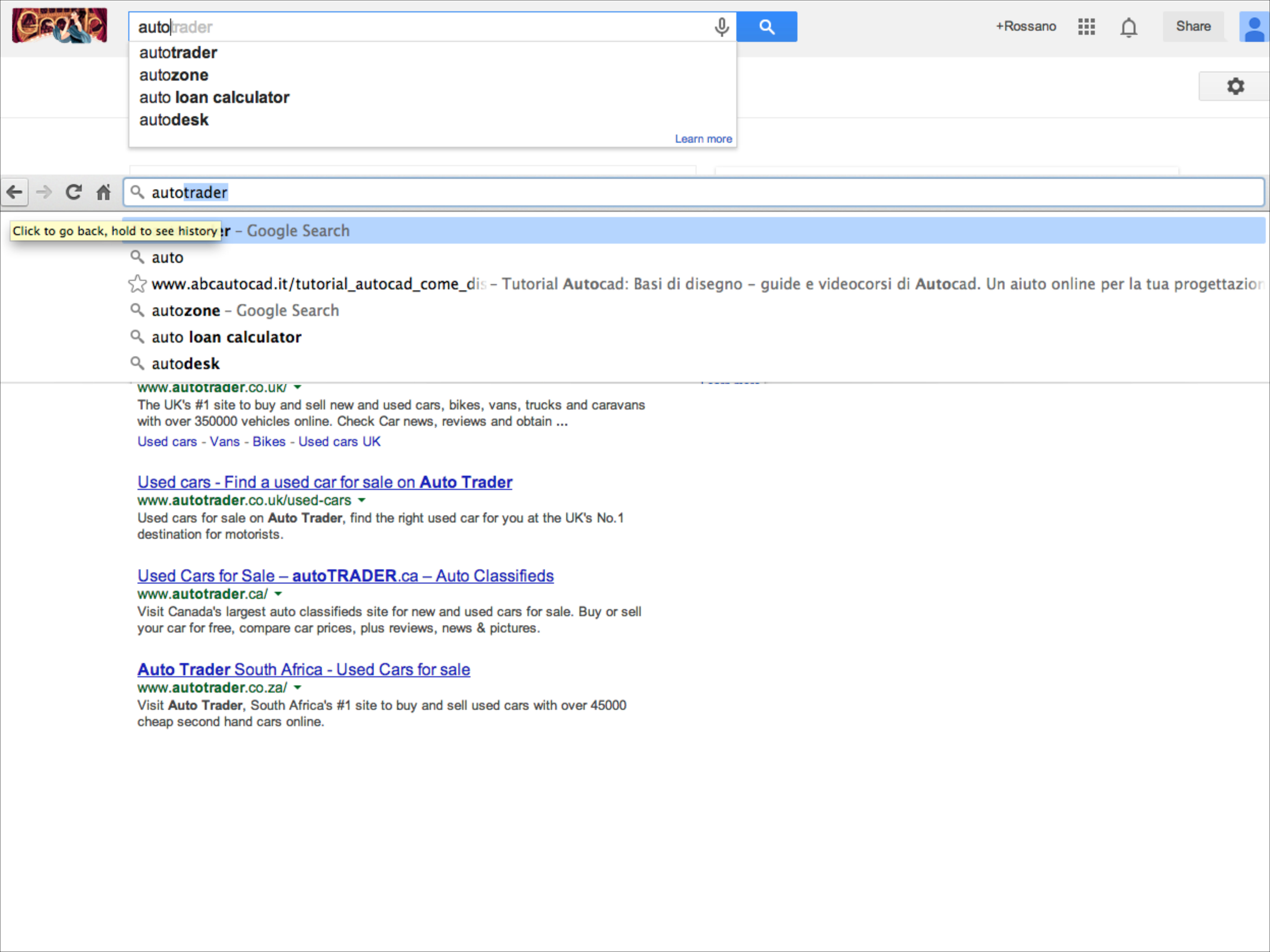
Used Cars for Sale – autoTRADER.ca – Auto Classifieds
www.autotrader.ca/
Visit Canada's largest auto classifieds site for new and used cars for sale. Buy or sell your car for free, compare car prices, plus reviews, news & pictures.

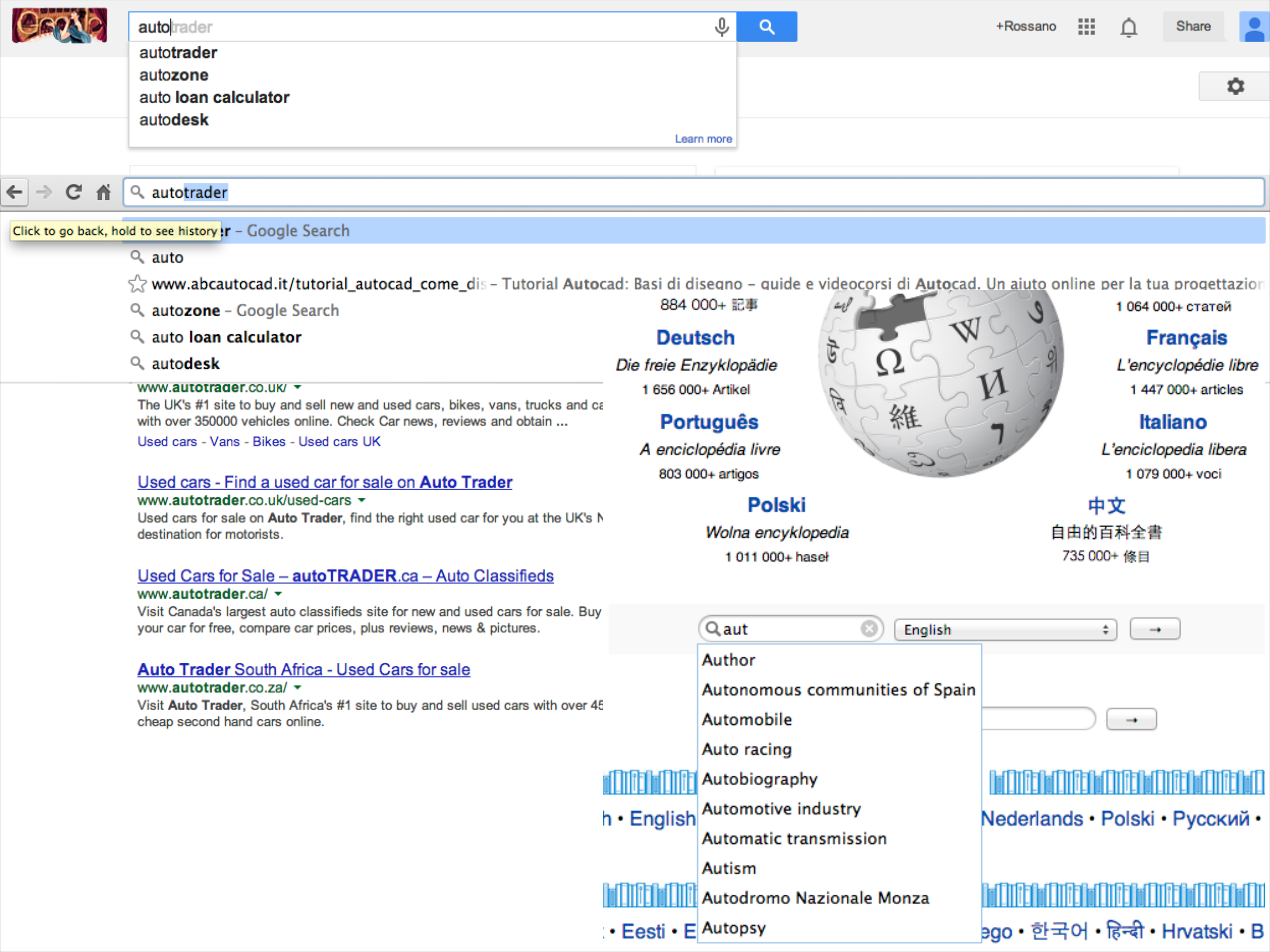
Auto Trader South Africa - Used Cars for sale
www.autotrader.co.za/
Visit **Auto Trader**, South Africa's #1 site to buy and sell used cars with over 45000 cheap second hand cars online.

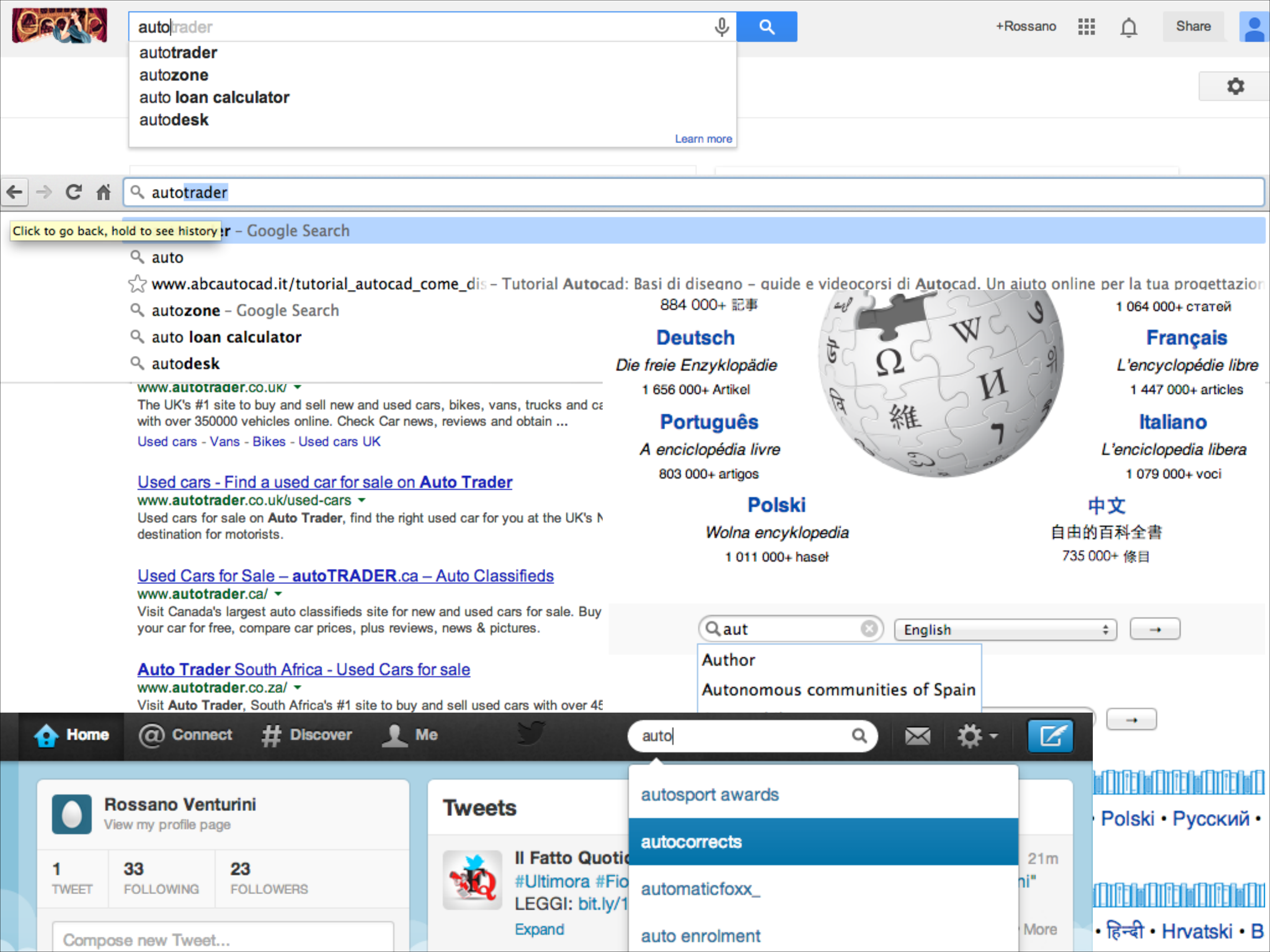
See results about

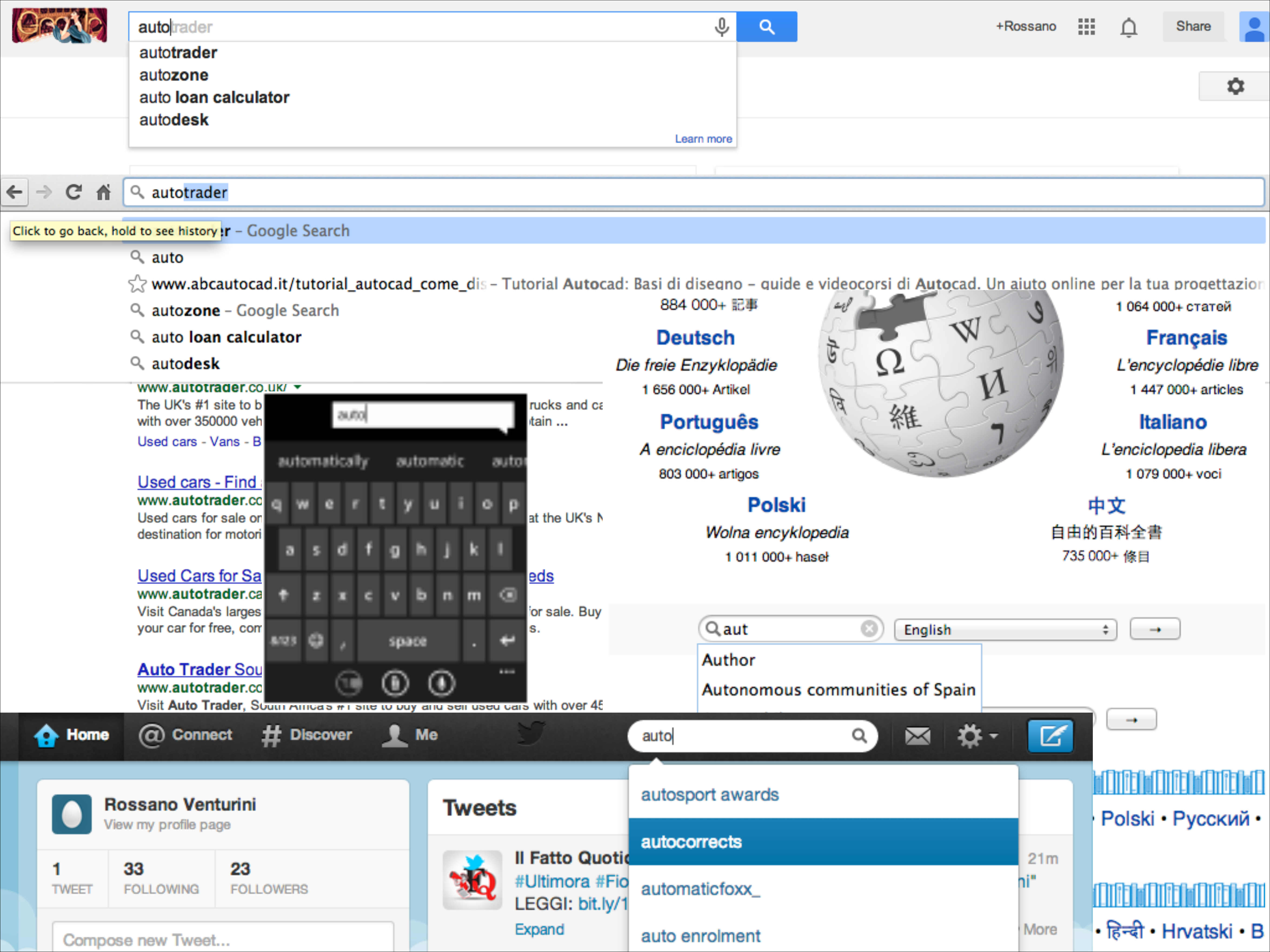
AutoTrader.com
Corporation
AutoTrader.com, Inc. is an online marketplace for car shoppers and sellers. It aggregates millions of new, ...

Feedback/More info


autotrader
+Rossano
Share
autotrader
autozone
auto loan calculator
autodesk
Learn more
autotrader
Click to go back, hold to see history
r – Google Search
auto
www.abcautocad.it/tutorial_autocad_come_dis – Tutorial Autocad: Basi di disegno – guide e videocorsi di Autocad. Un aiuto online per la tua progettazion
autozone – Google Search
auto loan calculator
autodesk
www.autotrader.co.uk/
The UK's #1 site to buy and sell new and used cars, bikes, vans, trucks and caravans with over 350000 vehicles online. Check Car news, reviews and obtain ...
Used cars - Vans - Bikes - Used cars UK
Used cars - Find a used car for sale on Auto Trader
www.autotrader.co.uk/used-cars
Used cars for sale on Auto Trader, find the right used car for you at the UK's No.1 destination for motorists.
Used Cars for Sale – autoTRADER.ca – Auto Classifieds
www.autotrader.ca/
Visit Canada's largest auto classifieds site for new and used cars for sale. Buy or sell your car for free, compare car prices, plus reviews, news & pictures.
Auto Trader South Africa - Used Cars for sale
www.autotrader.co.za/
Visit Auto Trader, South Africa's #1 site to buy and sell used cars with over 45000 cheap second hand cars online.


autotrader
autotrader
autozone
auto loan calculator
autodesk
Learn more
+Rossano
Share
autotrader
Click to go back, hold to see history
– Google Search
auto
www.abcautocad.it/tutorial_autocad_come_dis – Tutorial Autocad: Basi di disegno – guide e videocorsi di Autocad. Un aiuto online per la tua progettazion
autozone – Google Search
auto loan calculator
autodesk
www.autotrader.co.uk/
The UK's #1 site to buy and sell new and used cars, bikes, vans, trucks and ca with over 350000 vehicles online. Check Car news, reviews and obtain ...
Used cars - Vans - Bikes - Used cars UK
Used cars - Find a used car for sale on Auto Trader
www.autotrader.co.uk/used-cars
Used cars for sale on Auto Trader, find the right used car for you at the UK's N destination for motorists.
Used Cars for Sale – autoTRADER.ca – Auto Classifieds
www.autotrader.ca/
Visit Canada's largest auto classifieds site for new and used cars for sale. Buy your car for free, compare car prices, plus reviews, news & pictures.
Auto Trader South Africa - Used Cars for sale
www.autotrader.co.za/
Visit Auto Trader, South Africa's #1 site to buy and sell used cars with over 45 cheap second hand cars online.
884 000+ 記事
1 064 000+ статей
Deutsch
Die freie Enzyklopädie
1 656 000+ Artikel
Français
L'encyclopédie libre
1 447 000+ articles
Português
A enciclopédia livre
803 000+ artigos
Italiano
L'enciclopedia libera
1 079 000+ voci
Polski
Wolna encyklopedia
1 011 000+ haseł
中文
自由的百科全書
735 000+ 條目
aut
English
Author
Autonomous communities of Spain
Automobile
Auto racing
Autobiography
Automotive industry
Automatic transmission
Autism
Autodromo Nazionale Monza
Autopsy
h • English
Nederlands • Polski • Русский •
• Eesti • E
ego • 한국어 • हिन्दी • Hrvatski • B


auto|trader
autotrader
autozone
auto loan calculator
autodesk
Learn more
+Rossano
Share
autotrader
Click to go back, hold to see history
r – Google Search
auto
www.abcautocad.it/tutorial_autocad_come_dis – Tutorial Autocad: Basi di disegno – guide e videocorsi di Autocad. Un aiuto online per la tua progettazion
autozone – Google Search
auto loan calculator
autodesk
www.autotrader.co.uk/
The UK's #1 site to buy and sell new and used cars, bikes, vans, trucks and ca
with over 350000 vehicles online. Check Car news, reviews and obtain ...
Used cars - Vans - Bikes - Used cars UK
Used cars - Find a used car for sale on Auto Trader
www.autotrader.co.uk/used-cars
Used cars for sale on Auto Trader, find the right used car for you at the UK's N
destination for motorists.
Used Cars for Sale – autoTRADER.ca – Auto Classifieds
www.autotrader.ca/
Visit Canada's largest auto classifieds site for new and used cars for sale. Buy
your car for free, compare car prices, plus reviews, news & pictures.
Auto Trader South Africa - Used Cars for sale
www.autotrader.co.za/
Visit Auto Trader, South Africa's #1 site to buy and sell used cars with over 45
884 000+ 記事
1 064 000+ статей
Deutsch
Die freie Enzyklopädie
1 656 000+ Artikel
Français
L'encyclopédie libre
1 447 000+ articles
Português
A enciclopédia livre
803 000+ artigos
Italiano
L'enciclopedia libera
1 079 000+ voci
Polski
Wolna encyklopedia
1 011 000+ haseł
中文
自由的百科全書
735 000+ 條目
aut
English
Author
Autonomous communities of Spain
Home
Connect
Discover
Me
auto
autosport awards
autocorrects
automaticfoxx_
auto enrolment
Rossano Venturini
View my profile page
1 TWEET
33 FOLLOWING
23 FOLLOWERS
Compose new Tweet...
Tweets
Il Fatto Quotic
#Ultimora #Fio
LEGGI: bit.ly/1
Expand
21m
More
Polski • Русский •
हिन्दी • Hrvatski • B


autotrader
+Rossano
Share
autotrader
autozone
auto loan calculator
autodesk
Learn more
autotrader
Click to go back, hold to see history
– Google Search
auto
www.abcautocad.it/tutorial_autocad_come_di – Tutorial Autocad: Basi di disegno – guide e videocorsi di Autocad. Un aiuto online per la tua progettazion
autozone – Google Search
auto loan calculator
autodesk
www.autotrader.co.uk/
The UK's #1 site to b
with over 350000 veh
Used cars - Vans - B
Used cars - Find
www.autotrader.co
Used cars for sale or
destination for motori
Used Cars for Sa
www.autotrader.ca
Visit Canada's larges
your car for free, com
Auto Trader Sou
www.autotrader.co
Visit Auto Trader, South Africa's #1 site to buy and sell used cars with over 45
auto
automatically
automatic
space
884 000+ 記事
1 064 000+ статей
Deutsch
Die freie Enzyklopädie
1 656 000+ Artikel
Français
L'encyclopédie libre
1 447 000+ articles
Português
A enciclopédia livre
803 000+ artigos
Italiano
L'enciclopedia libera
1 079 000+ voci
Polski
Wolna encyklopedia
1 011 000+ haseł
中文
自由的百科全書
735 000+ 條目
aut
English
Author
Autonomous communities of Spain
Home
Connect
Discover
Me
auto
Rossano Venturini
View my profile page
1 TWEET
33 FOLLOWING
23 FOLLOWERS
Compose new Tweet...
Tweets
Il Fatto Quoti
#Ultimora #Fio
LEGGI: bit.ly/1
Expand
21m
More
autosport awards
autocorrects
automaticfoxx_
auto enrolment
Polski • Русский •
हिन्दी • Hrvatski • B


Google!
BETA
Search the web using Google!
Google Search
I'm feeling lucky
Special Searches
Stanford Search
Linux Search
Why use Google!
Press about Google!
Help!
Company Info
Jobs at Google
Google! Logos
Making Google! the Default
Get Google!
updates monthly:
your e-mail
Subscribe
Archive
Copyright ©1999 Google Inc.

Google!
BETA
Search the web using Google!
Google Search
I'm feeling lucky
Special Searches
Stanford Search
Linux Search
Why use Google!
Press about Google!
Help!
Company Info
Jobs at Google
Google! Logos
Making Google! the Default
Get Google!
updates monthly:
your e-mail
Subscribe
Archive
Copyright ©1999 Google Inc.

Dataset?

Dataset?

All the past queries

Dataset?

Searches?

All the past queries

Dataset?

Searches?

All the past queries

Prefix search

Dataset?

Searches?

Data structure?

All the past queries

Prefix search

Dataset? All the past queries

Searches? Prefix search

Data structure? Trie

Dataset?	All the past queries
Searches?	Prefix search
Data structure?	Trie

How to find top-k efficiently?

Trie

Trie

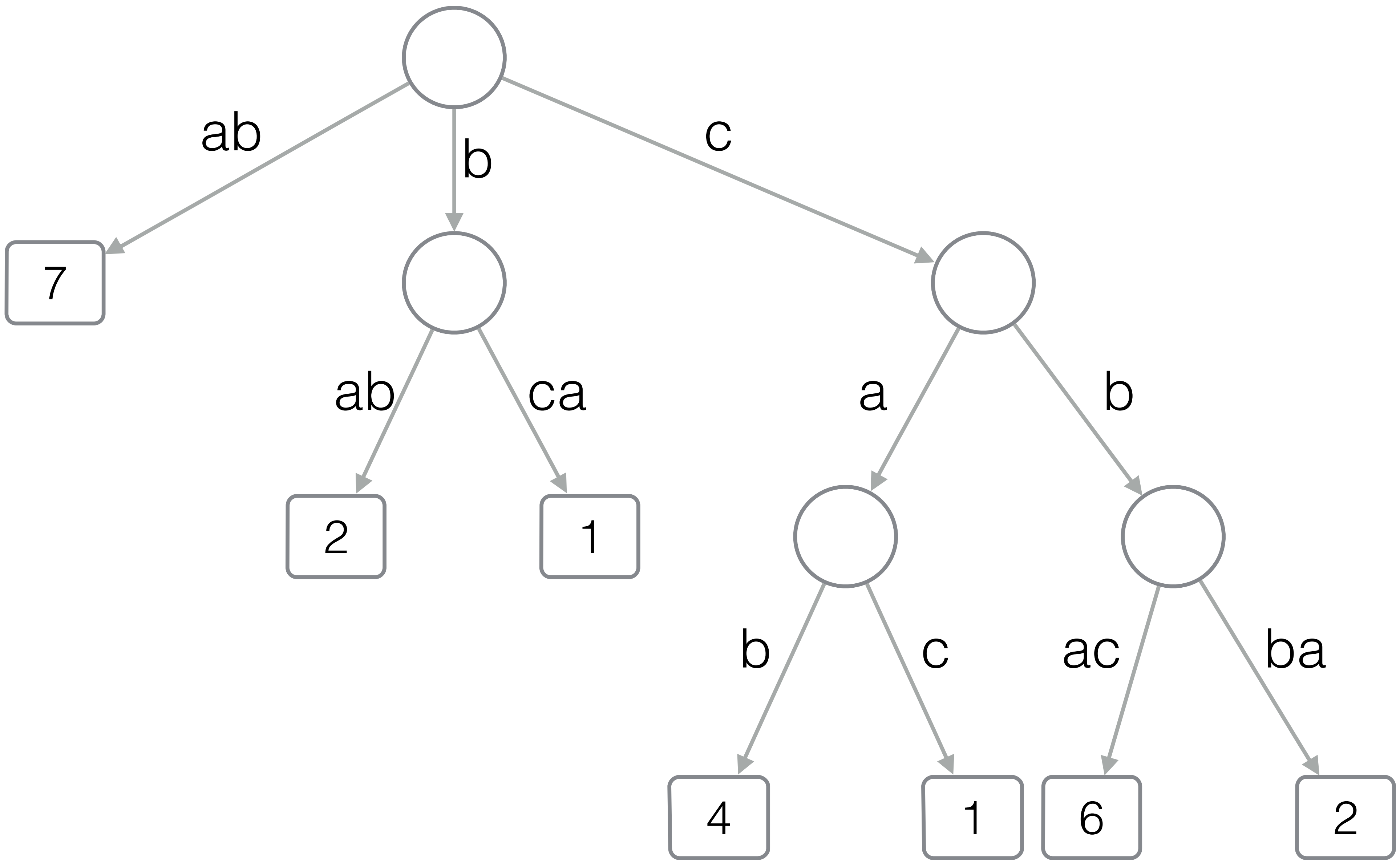
D = { ab (7), bab (2), bca (1), cab (4), cac (1), cbac (6), cbba (2) }

Trie

D = { ab (7), bab (2), bca (1), cab (4), cac (1), cbac (6), cbba (2) }

n = |D|, m total length of strings in D

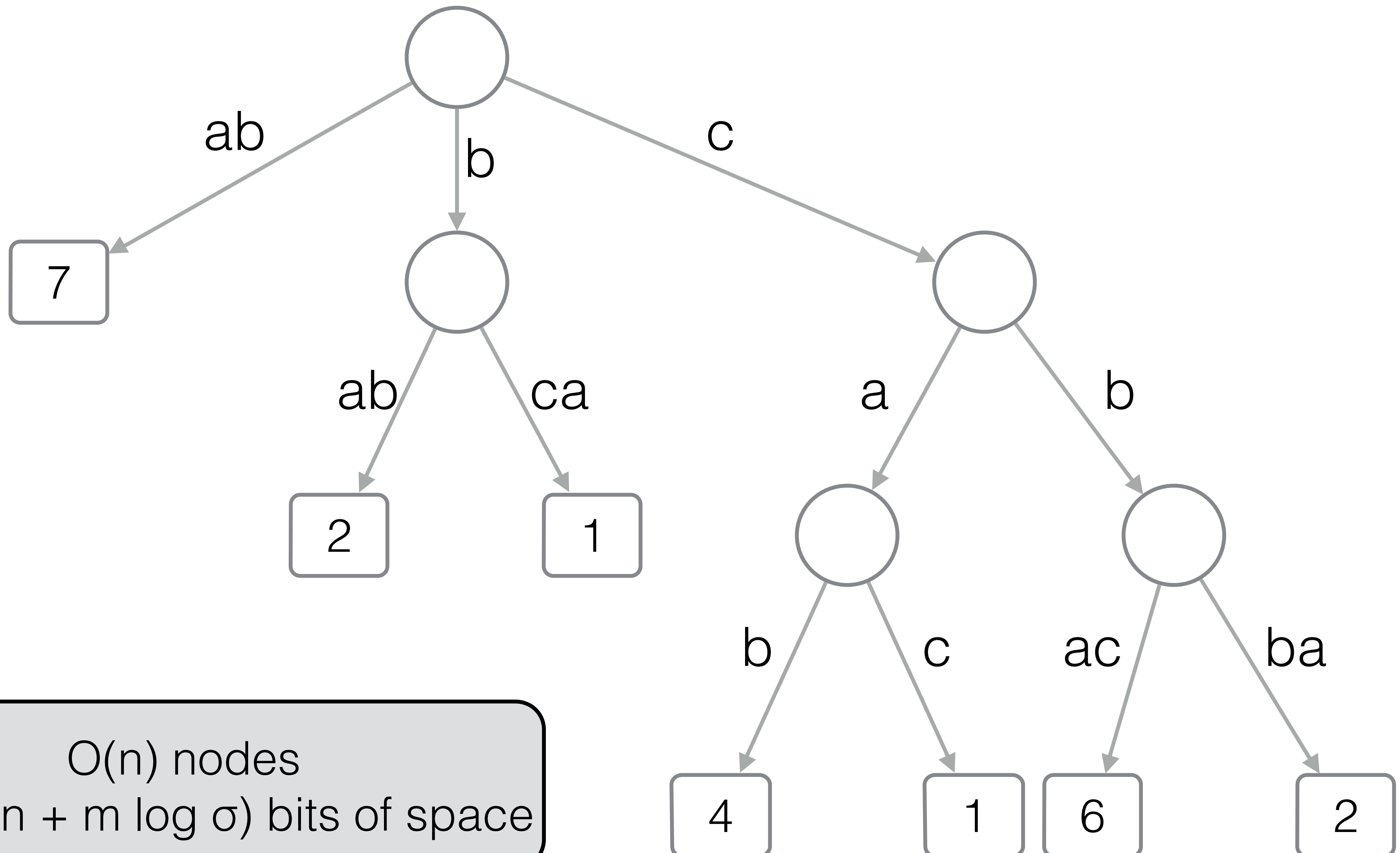
Trie



D = { ab (7), bab (2), bca (1), cab (4), cac (1), cbac (6), cbba (2) }

n = |D|, m total length of strings in D

Trie

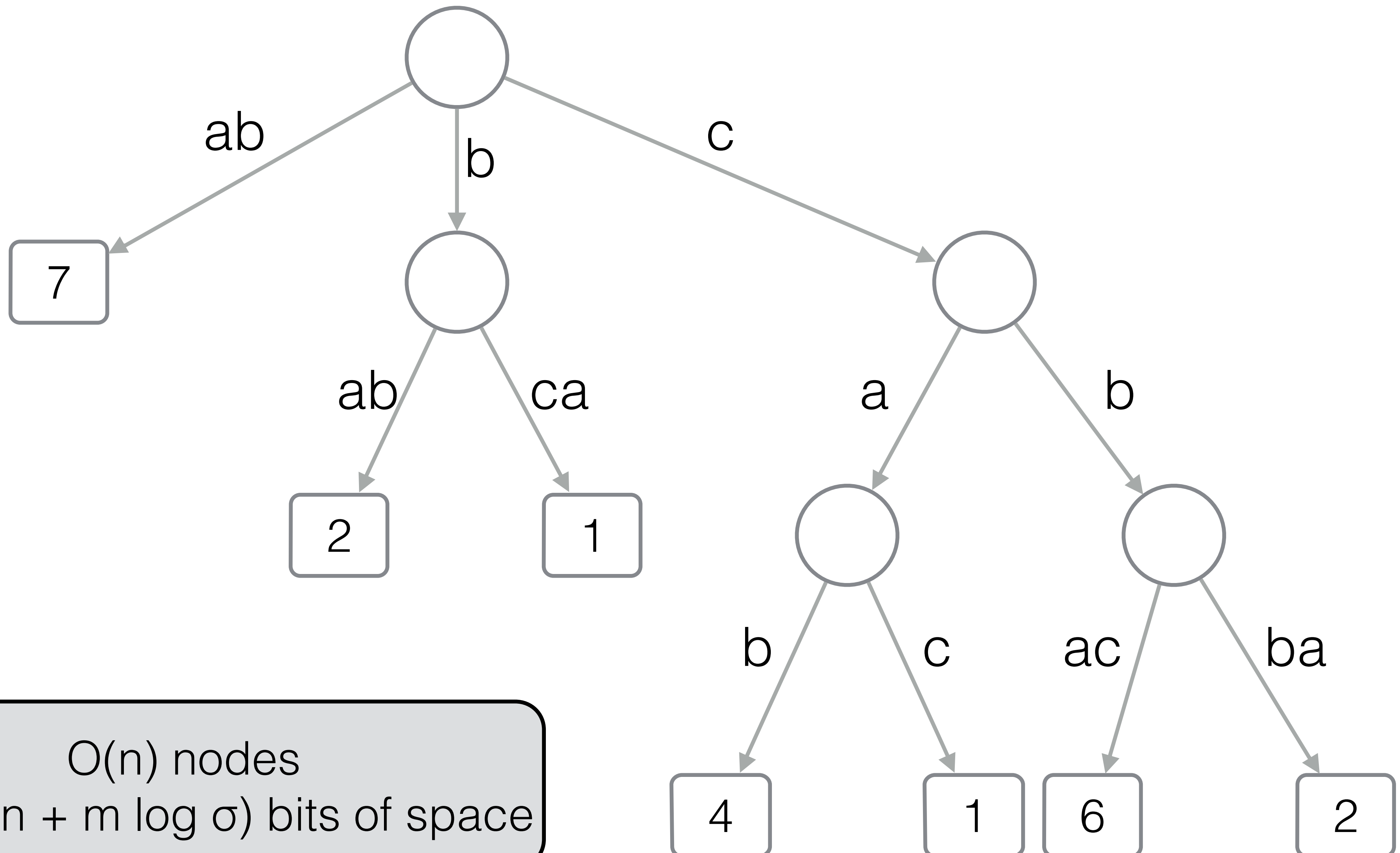


O(n) nodes
O(n log n + m log σ) bits of space

D = { ab (7), bab (2), bca (1), cab (4), cac (1), cbac (6), cbba (2) }

n = |D|, m total length of strings in D

Trie



O(n) nodes
O(n log n + m log σ) bits of space

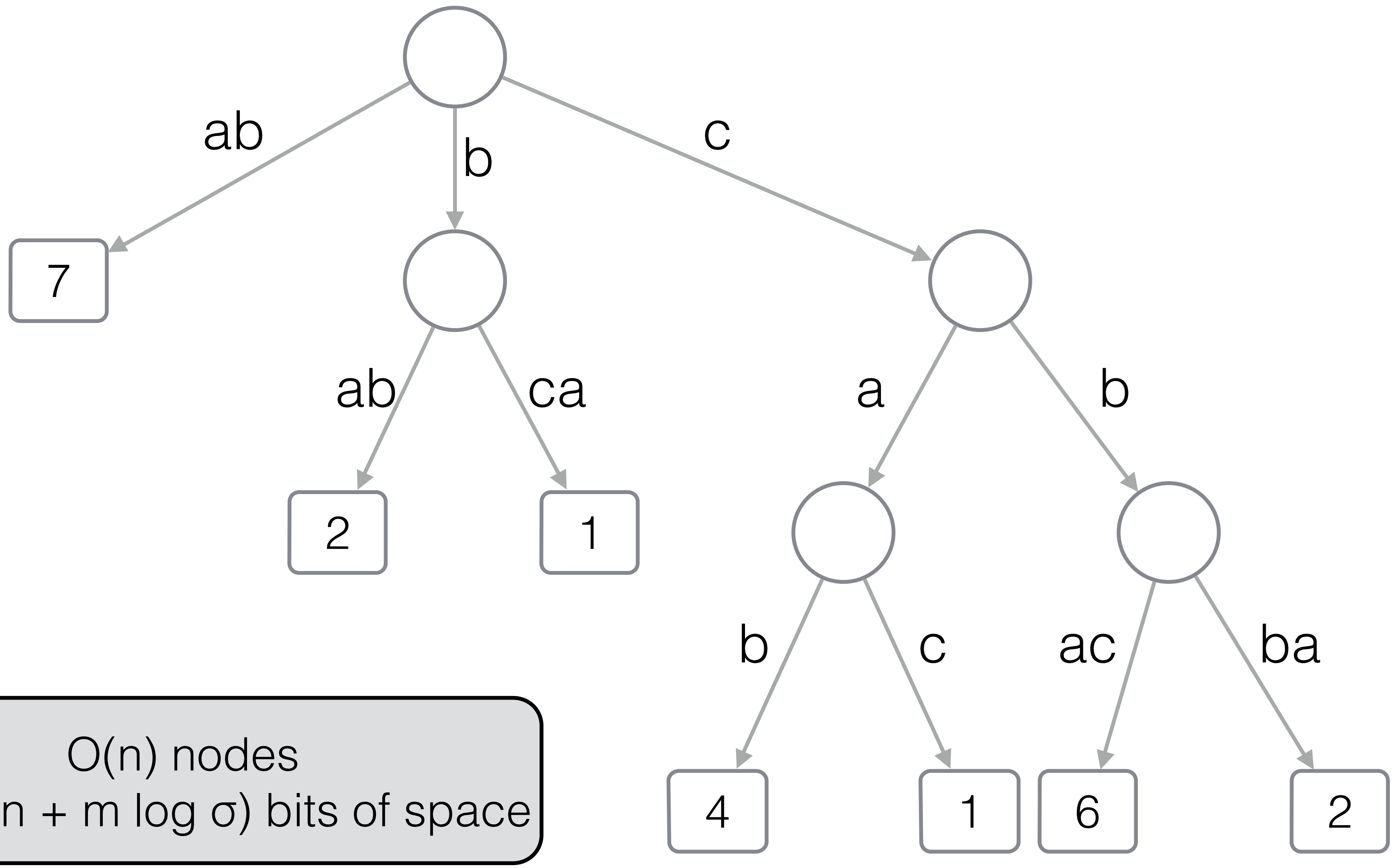
Find all the strings prefixed by any pattern P in O(|P|) time

D = { ab (7), bab (2), bca (1), cab (4), cac (1), cbac (6), cbba (2) }

n = |D|, m total length of strings in D

Trie

P = c



O(n) nodes
O(n log n + m log σ) bits of space

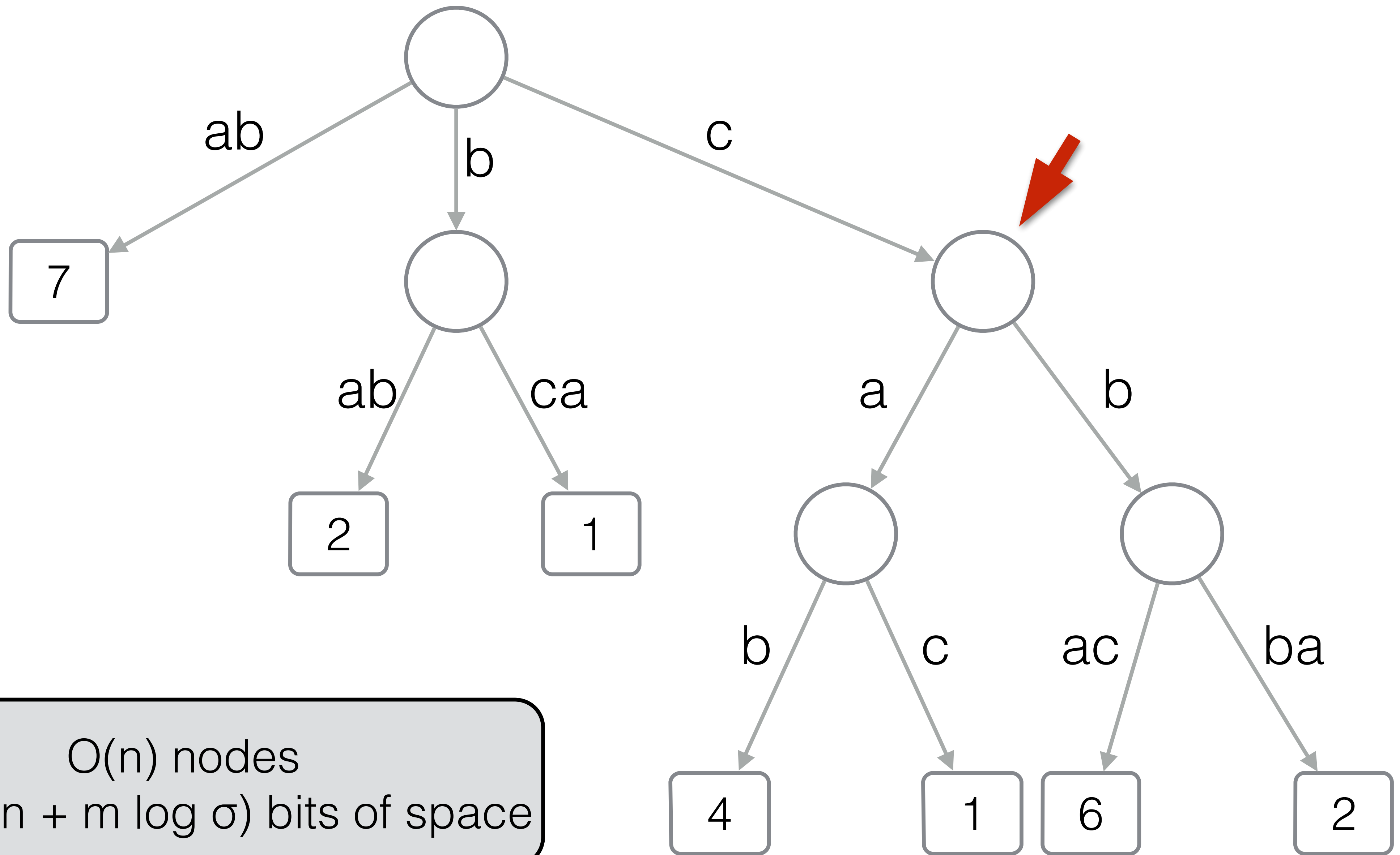
Find all the strings prefixed by any pattern P in O(|P|) time

D = { ab (7), bab (2), bca (1), cab (4), cac (1), cbac (6), cbba (2) }

n = |D|, m total length of strings in D

Trie

P = c



O(n) nodes
O(n log n + m log σ) bits of space

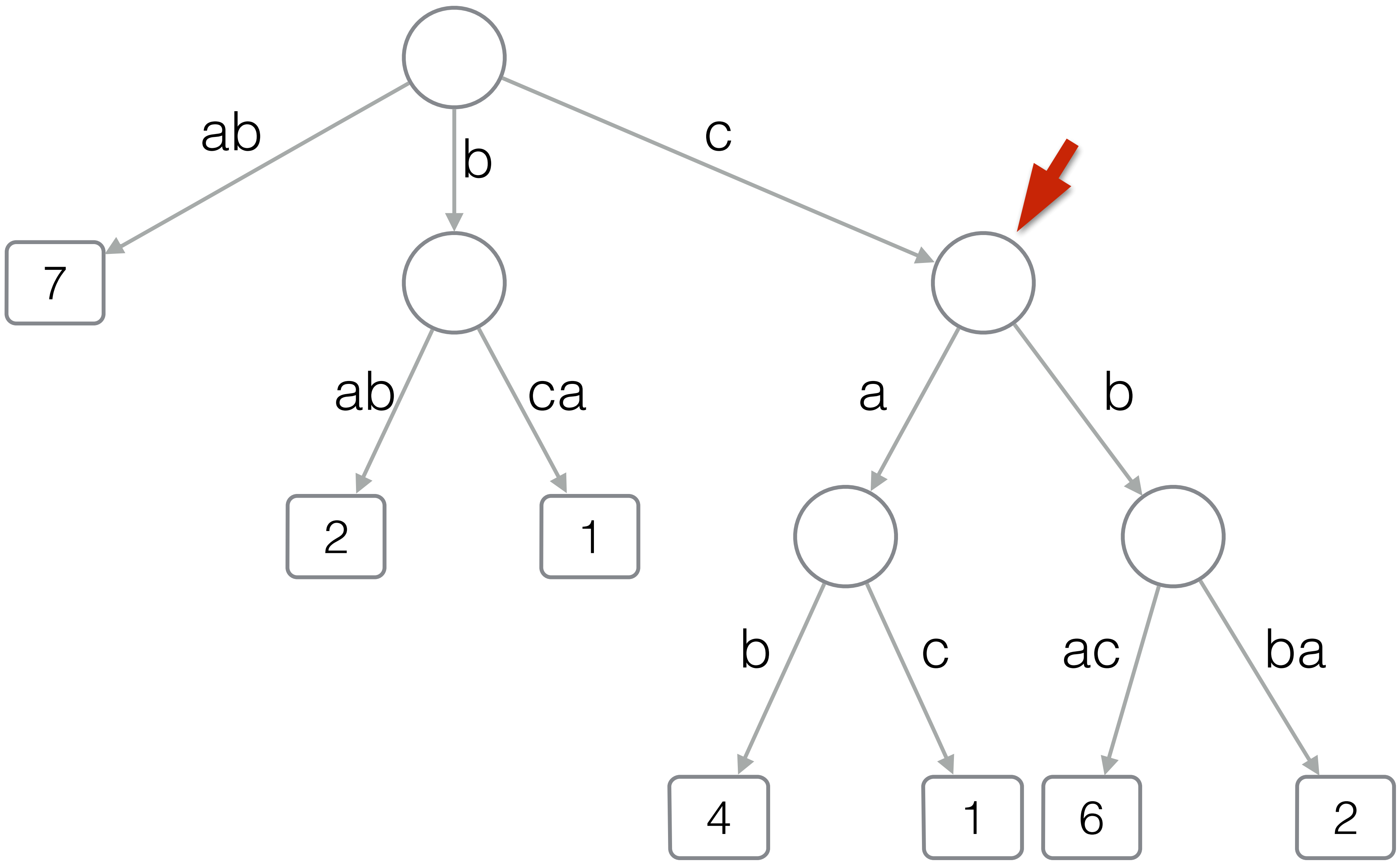
Find all the strings prefixed by any pattern P in O(|P|) time

D = { ab (7), bab (2), bca (1), cab (4), cac (1), cbac (6), cbba (2) }

n = |D|, m total length of strings in D

Finding Top-1

P = c



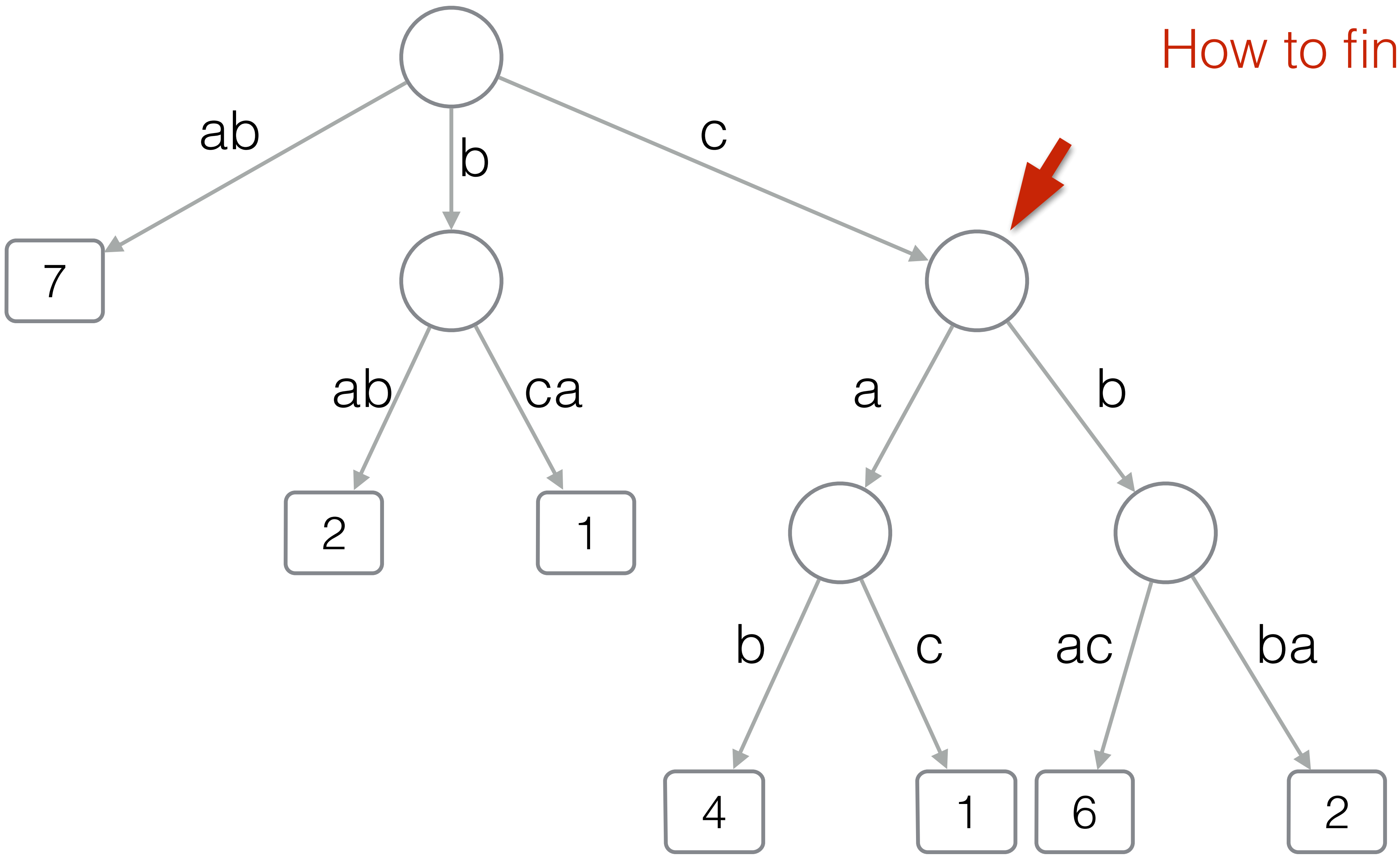
D = { ab (7), bab (2), bca (1), cab (4), cac (1), cbac (6), cbba (2) }

n = |D|, m total length of strings in D

Finding Top-1

P = c

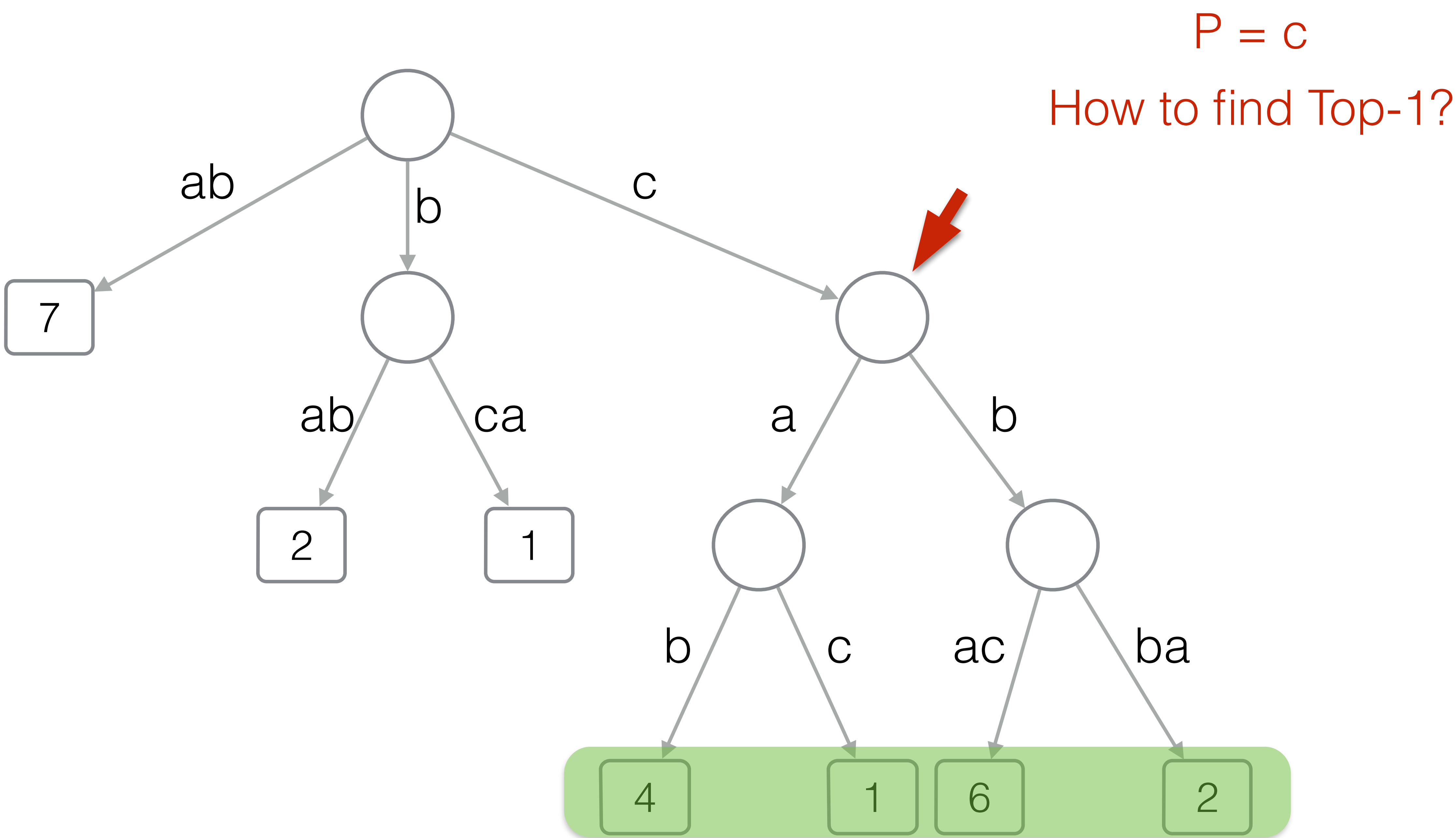
How to find Top-1?



D = { ab (7), bab (2), bca (1), cab (4), cac (1), cbac (6), cbba (2) }

n = |D|, m total length of strings in D

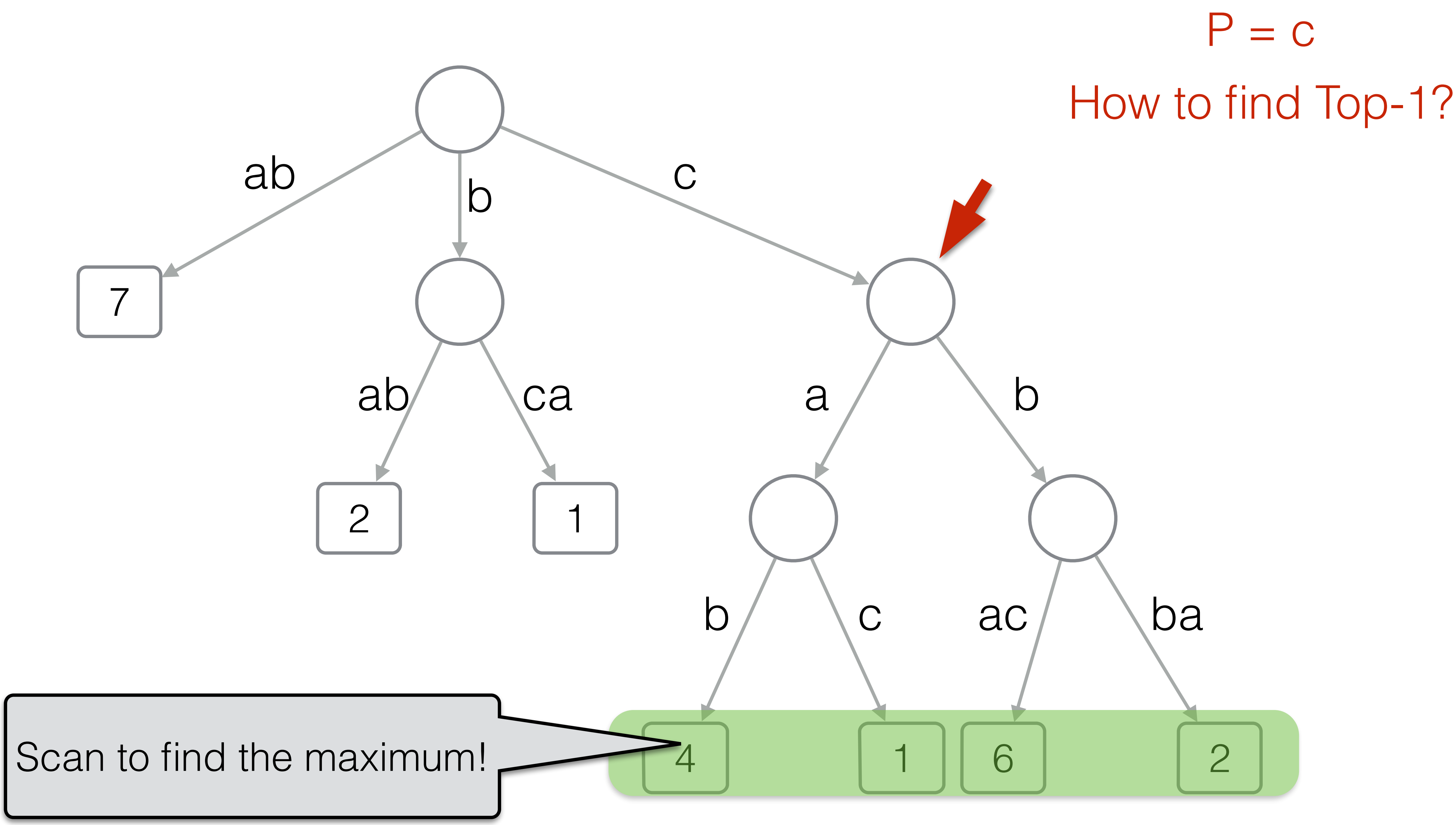
Finding Top-1



D = { ab (7), bab (2), bca (1), cab (4), cac (1), cbac (6), cbba (2) }

n = |D|, m total length of strings in D

Finding Top-1



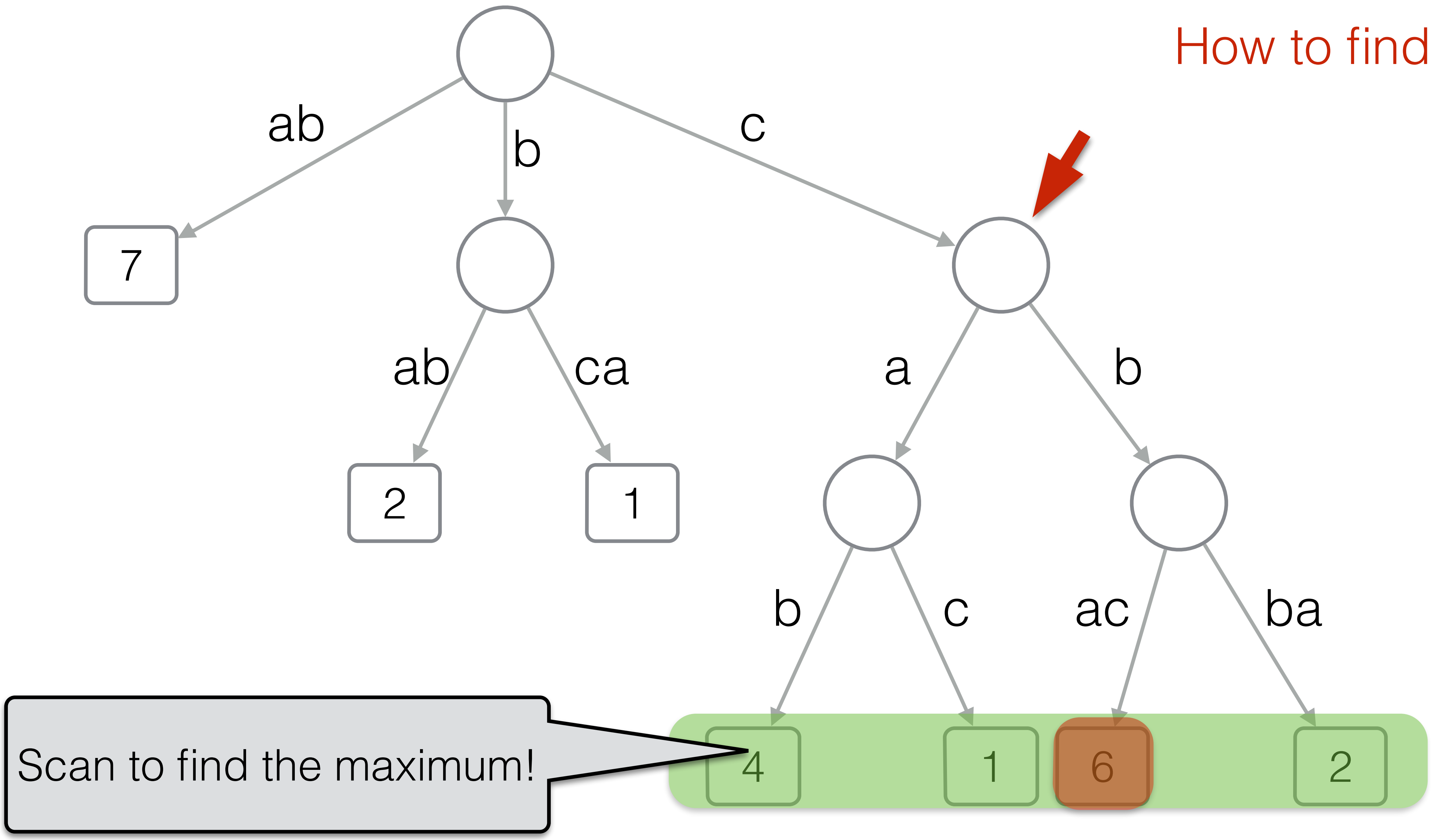
D = { ab (7), bab (2), bca (1), cab (4), cac (1), cbac (6), cbba (2) }

n = |D|, m total length of strings in D

Finding Top-1

P = c

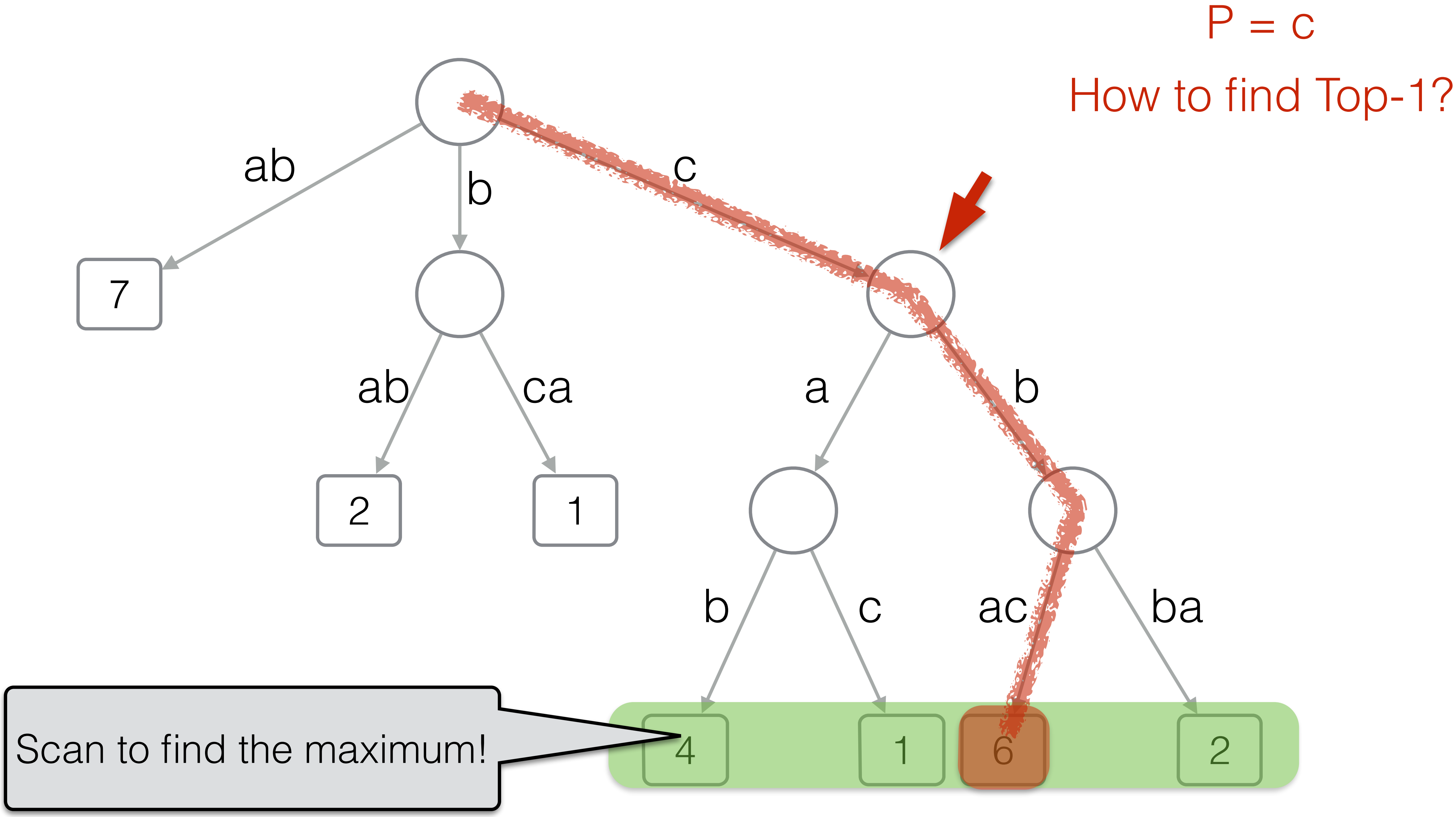
How to find Top-1?



D = { ab (7), bab (2), bca (1), cab (4), cac (1), cbac (6), cbba (2) }

n = |D|, m total length of strings in D

Finding Top-1



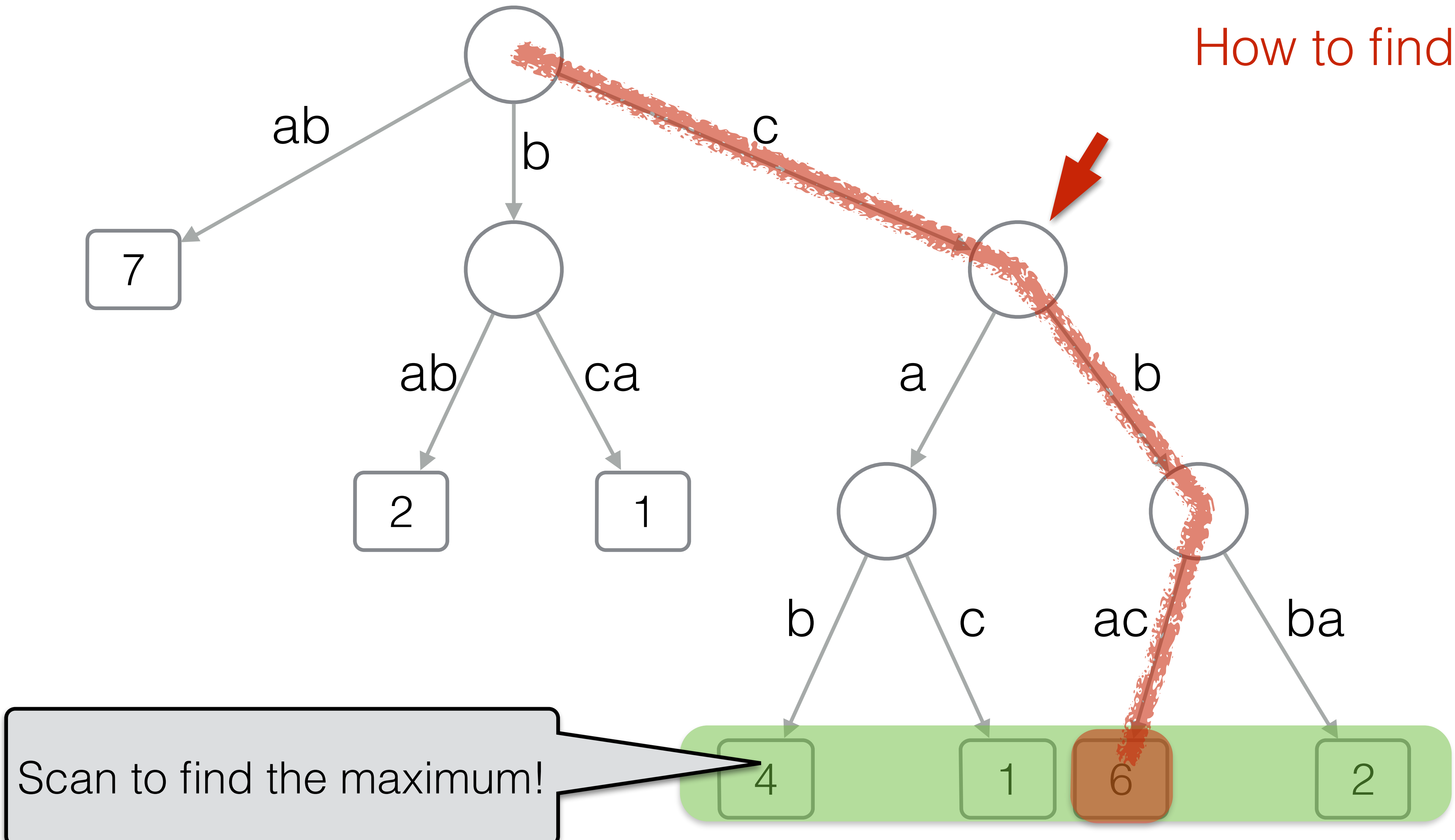
D = { ab (7), bab (2), bca (1), cab (4), cac (1), cbac (6), cbba (2) }

n = |D|, m total length of strings in D

Finding Top-1

P = c

How to find Top-1?



O(n) query time :-(

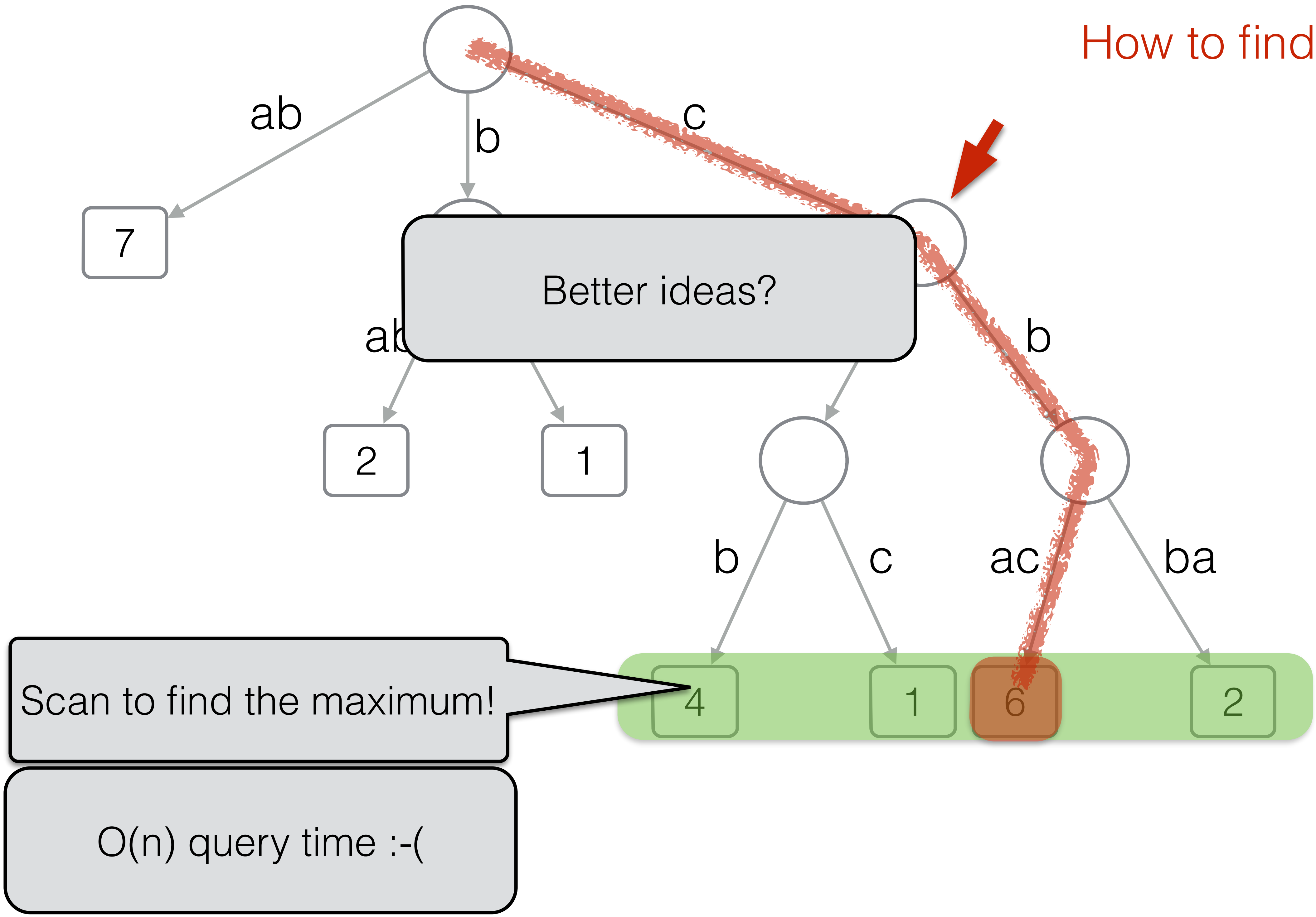
D = { ab (7), bab (2), bca (1), cab (4), cac (1), cbac (6), cbba (2) }

n = |D|, m total length of strings in D

Finding Top-1

P = c

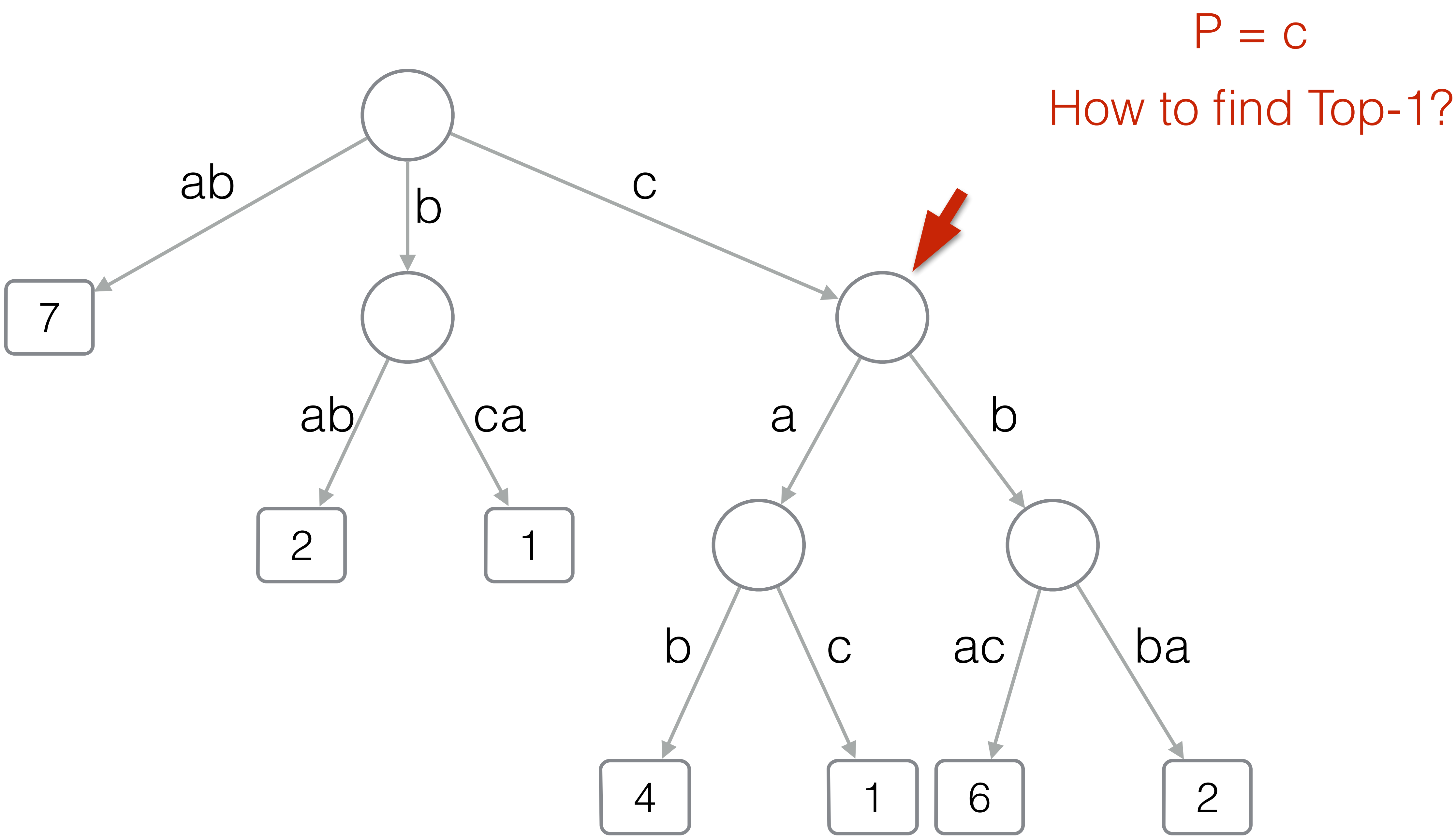
How to find Top-1?



D = { ab (7), bab (2), bca (1), cab (4), cac (1), cbac (6), cbba (2) }

n = |D|, m total length of strings in D

Finding Top-1



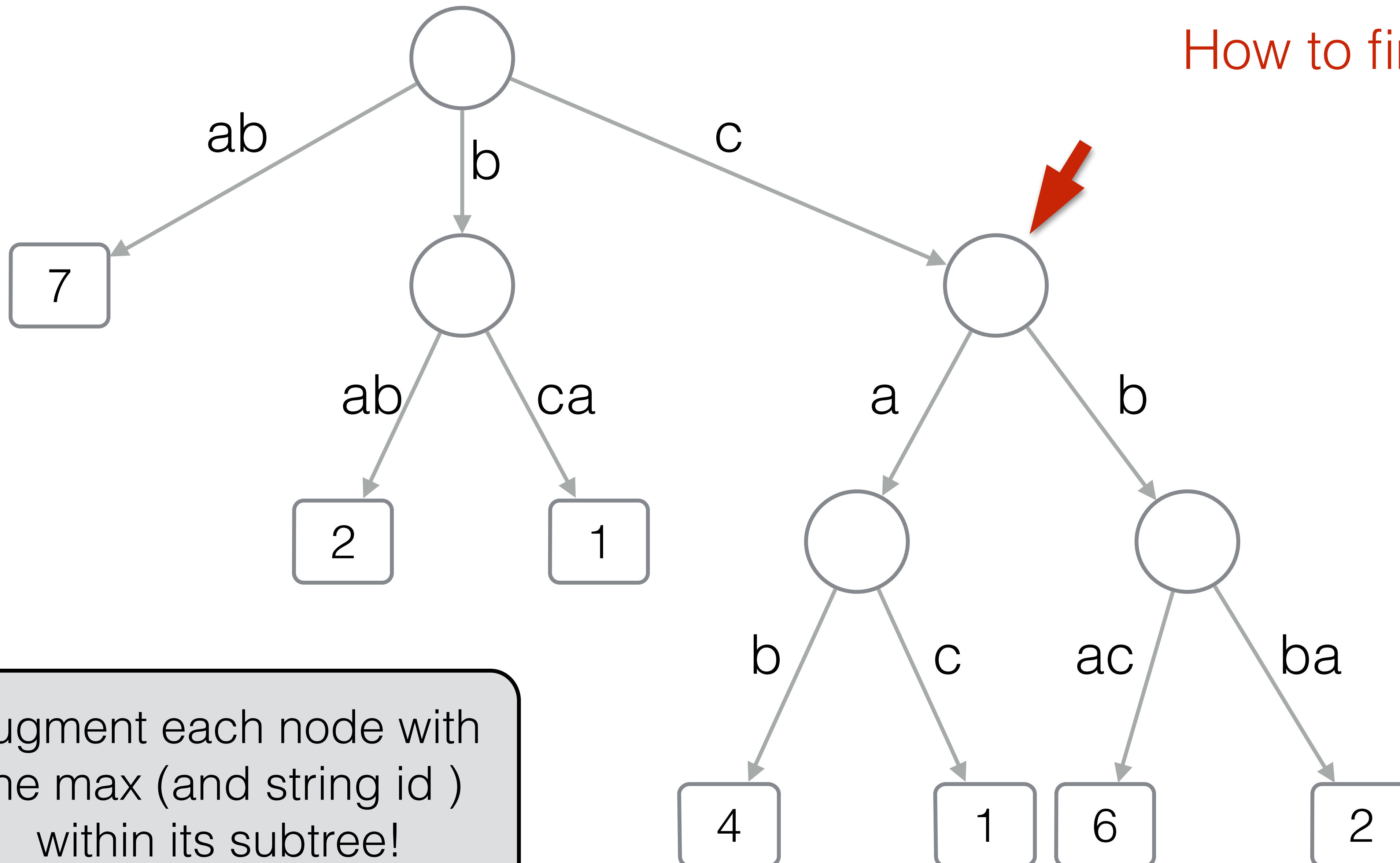
D = { ab (7), bab (2), bca (1), cab (4), cac (1), cbac (6), cbba (2) }

n = |D|, m total length of strings in D

Finding Top-1

P = c

How to find Top-1?



Augment each node with the max (and string id) within its subtree!

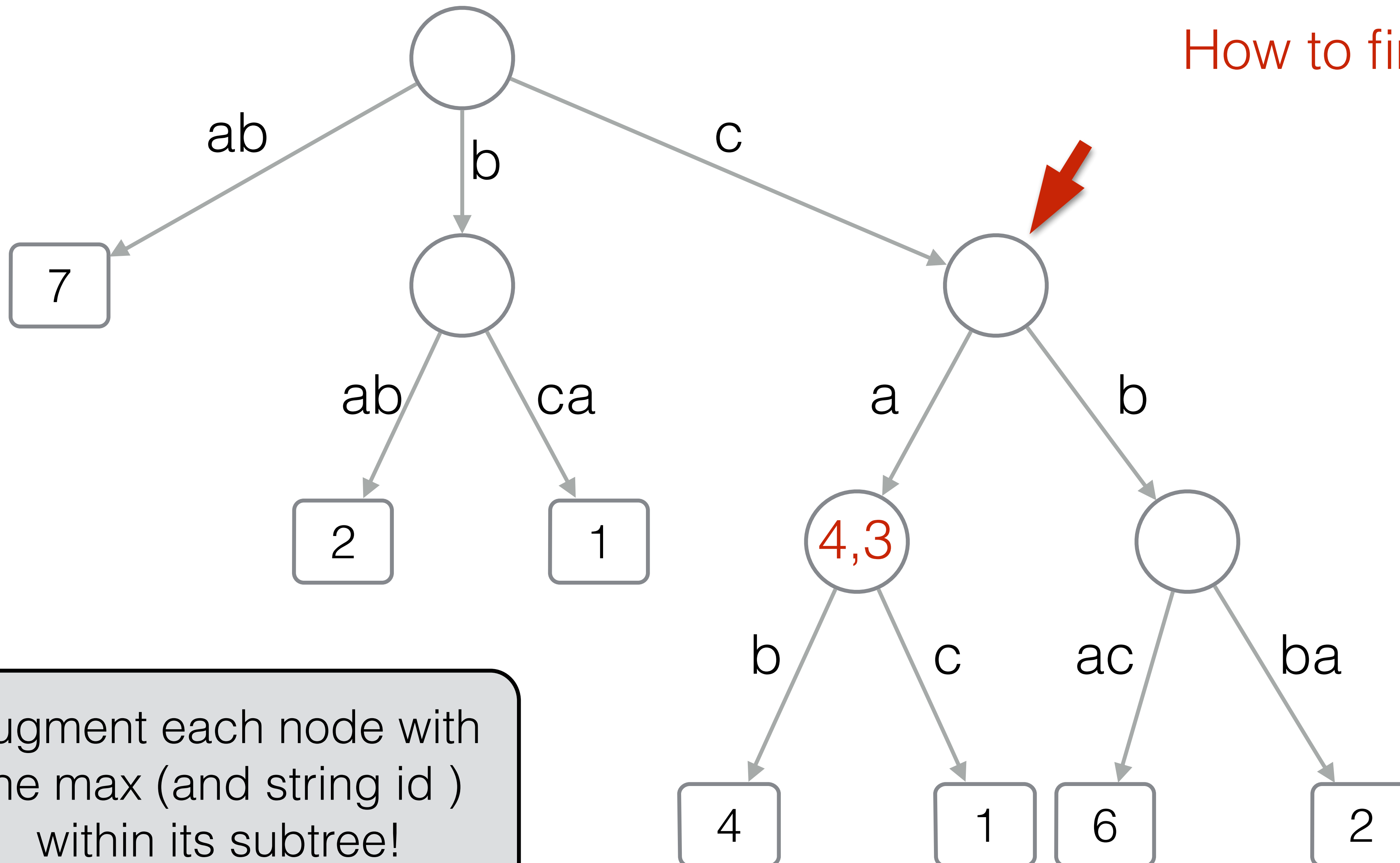
D = { ab (7), bab (2), bca (1), cab (4), cac (1), cbac (6), cbba (2) }

n = |D|, m total length of strings in D

Finding Top-1

P = c

How to find Top-1?



Augment each node with the max (and string id) within its subtree!

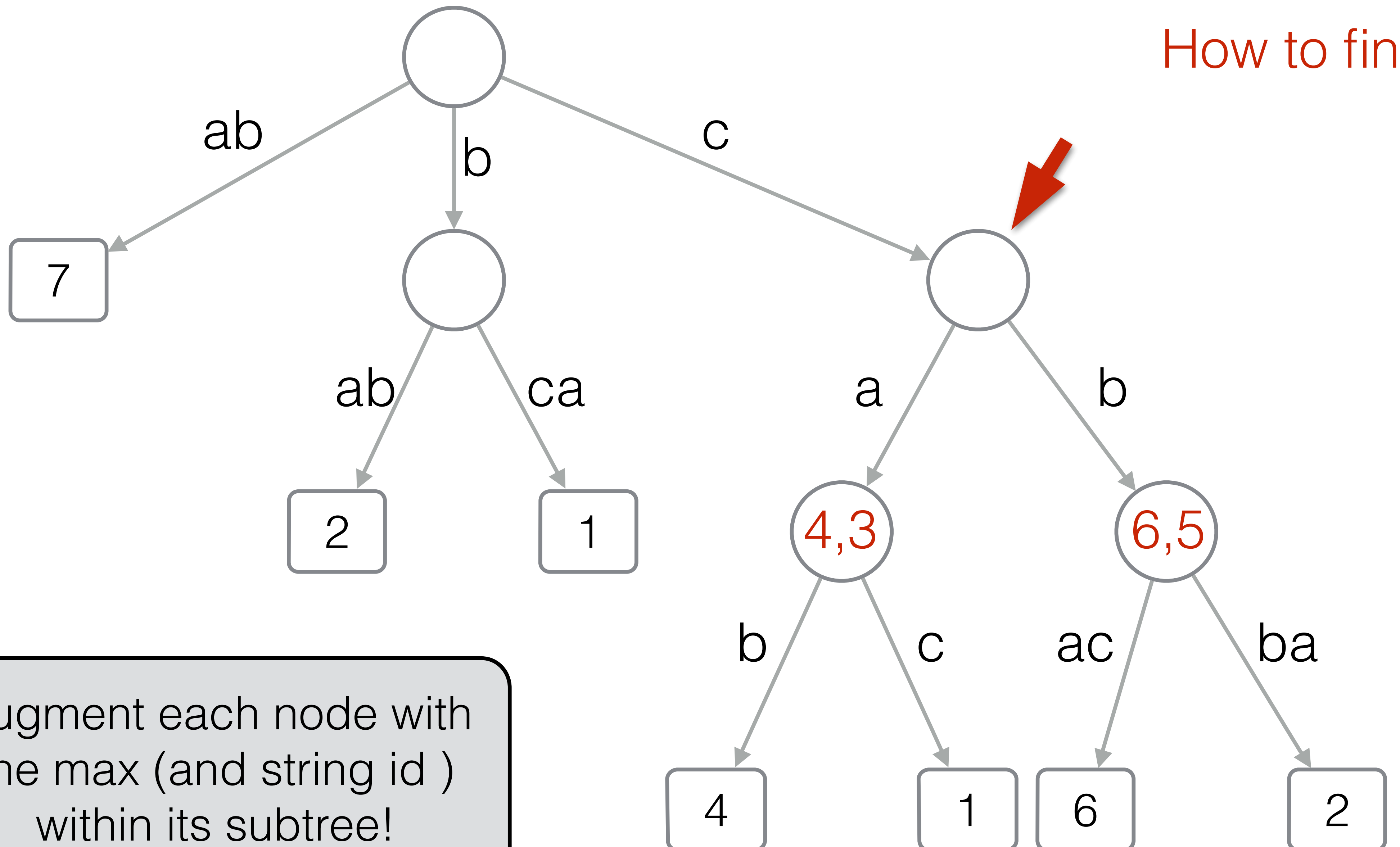
D = { ab (7), bab (2), bca (1), cab (4), cac (1), cbac (6), cbba (2) }

n = |D|, m total length of strings in D

Finding Top-1

P = c

How to find Top-1?

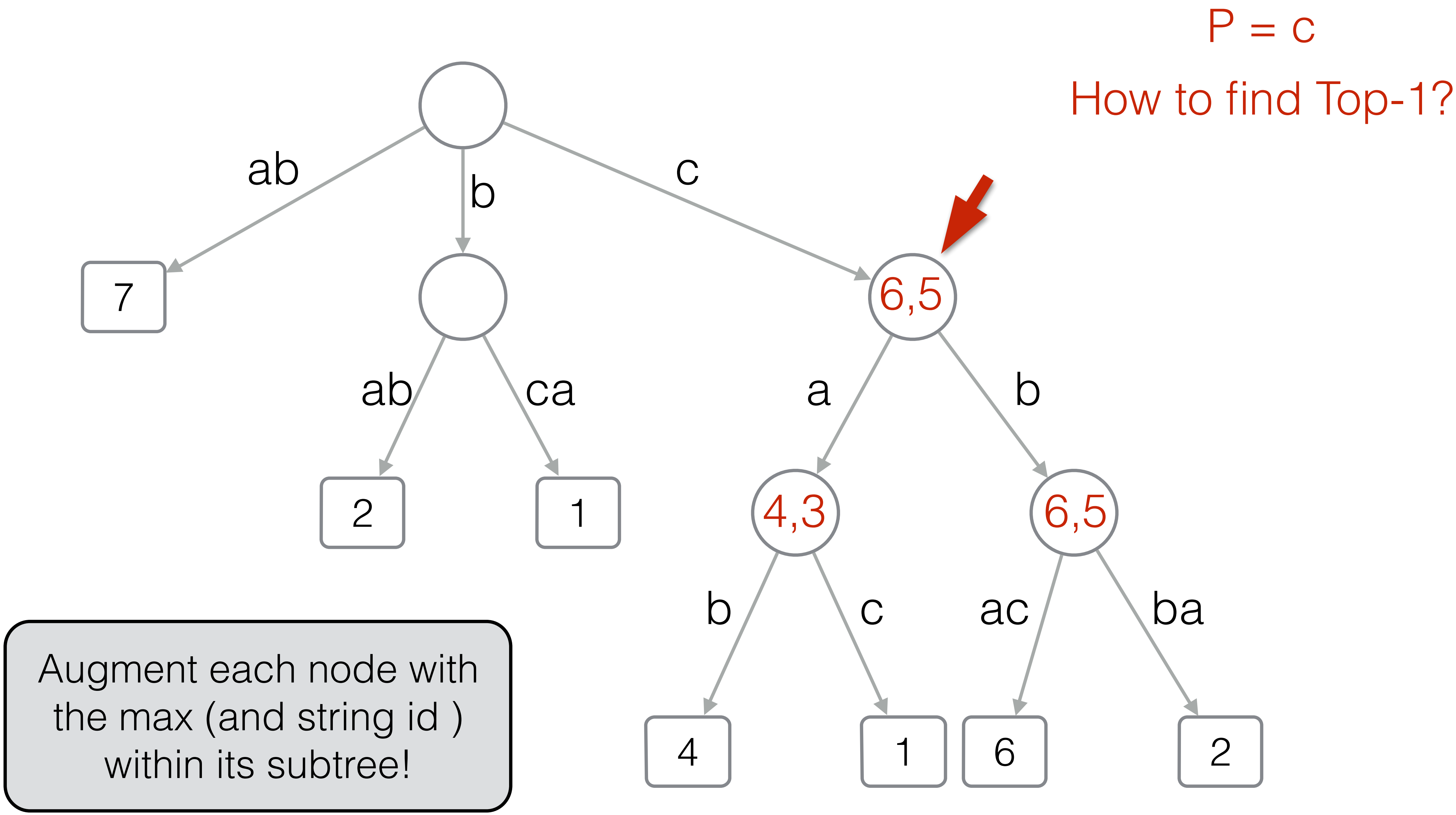


Augment each node with the max (and string id) within its subtree!

D = { ab (7), bab (2), bca (1), cab (4), cac (1), cbac (6), cbba (2) }

n = |D|, m total length of strings in D

Finding Top-1



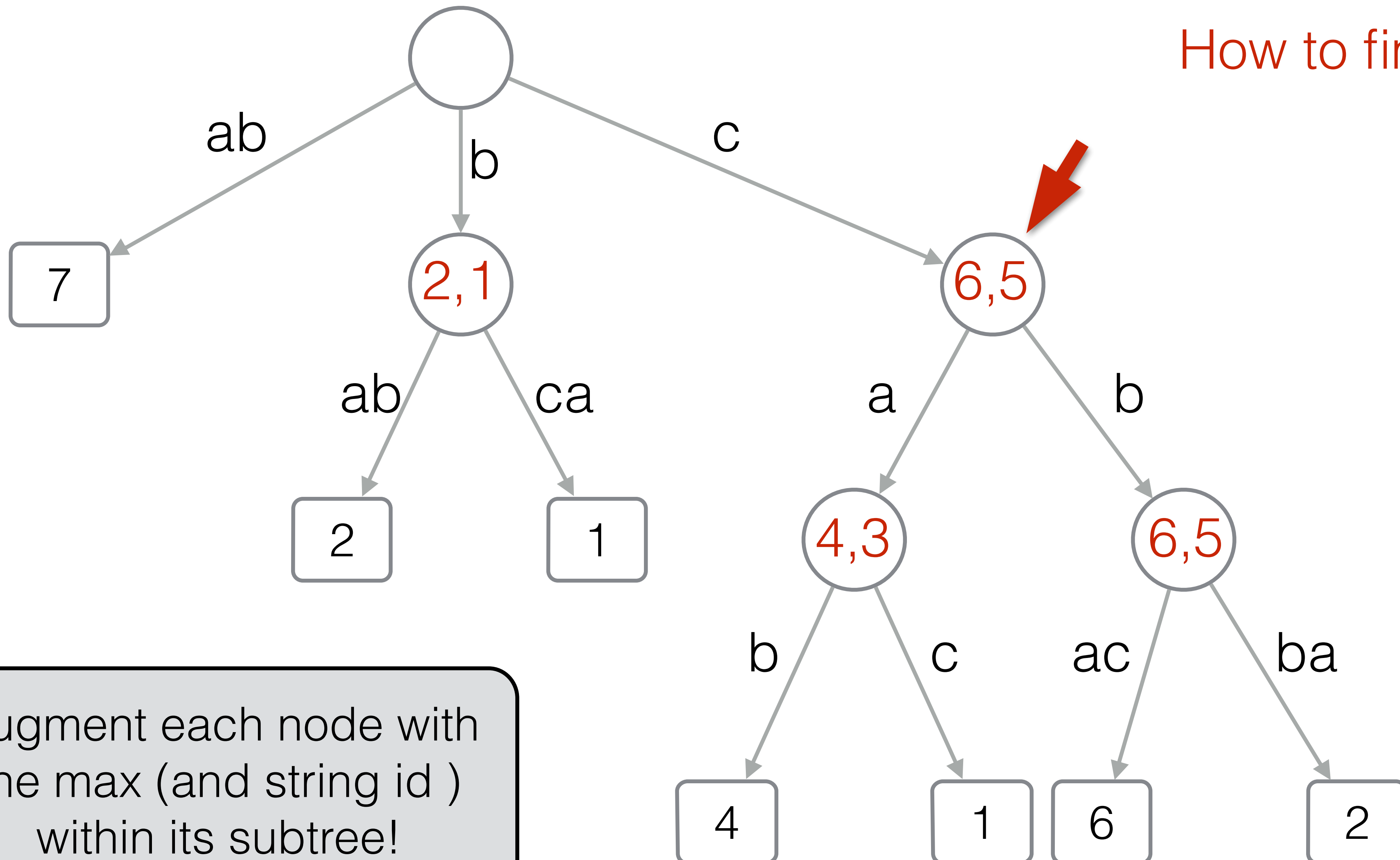
D = { ab (7), bab (2), bca (1), cab (4), cac (1), cbac (6), cbba (2) }

n = |D|, m total length of strings in D

Finding Top-1

P = c

How to find Top-1?



Augment each node with the max (and string id) within its subtree!

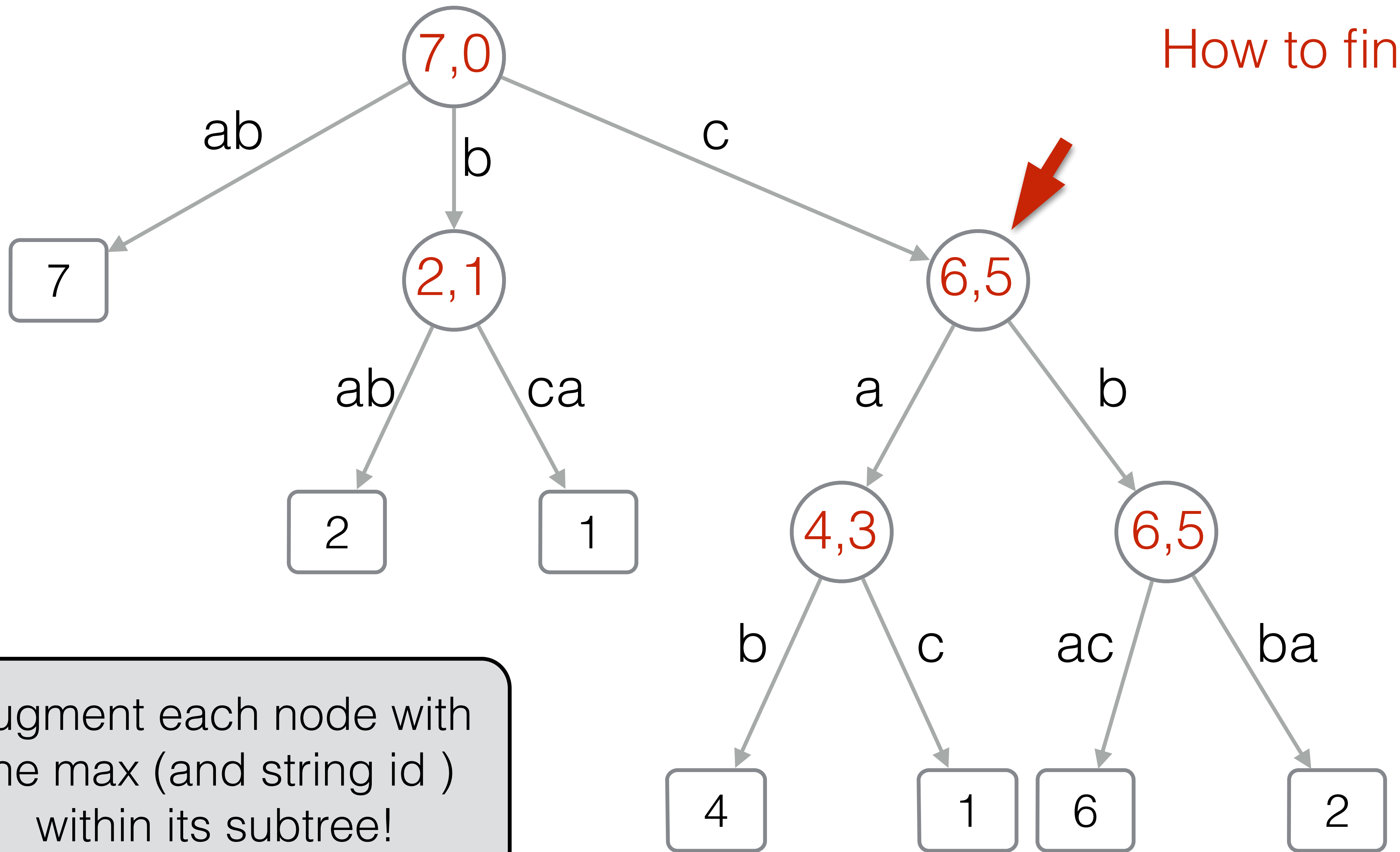
D = { ab (7), bab (2), bca (1), cab (4), cac (1), cbac (6), cbba (2) }

n = |D|, m total length of strings in D

Finding Top-1

P = c

How to find Top-1?



Augment each node with the max (and string id) within its subtree!

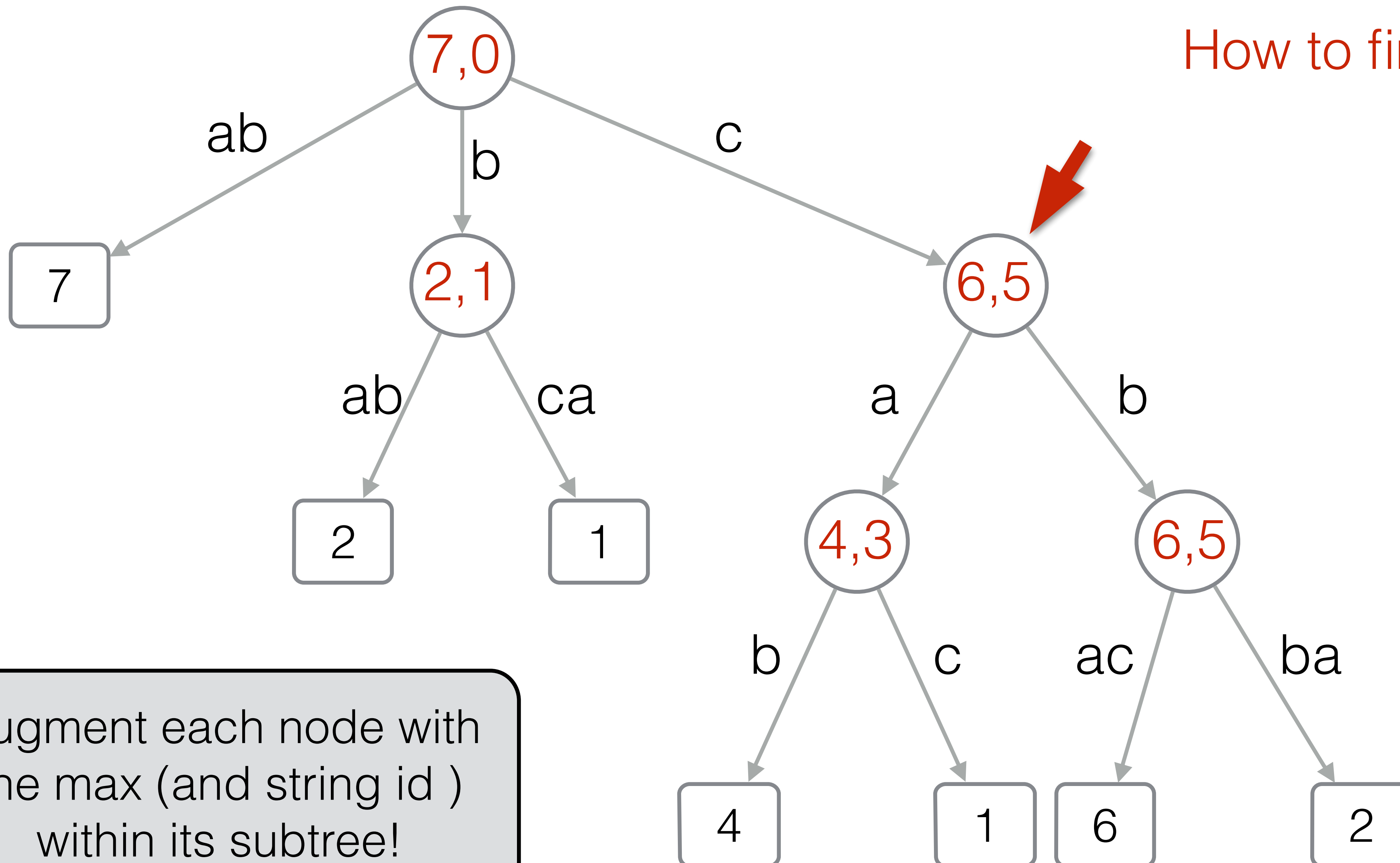
D = { ab (7), bab (2), bca (1), cab (4), cac (1), cbac (6), cbba (2) }

n = |D|, m total length of strings in D

Finding Top-1

P = c

How to find Top-1?



Augment each node with the max (and string id) within its subtree!

Preprocessing time: O(n)

Extra space: O(n log n) bits

Query time: O(1)

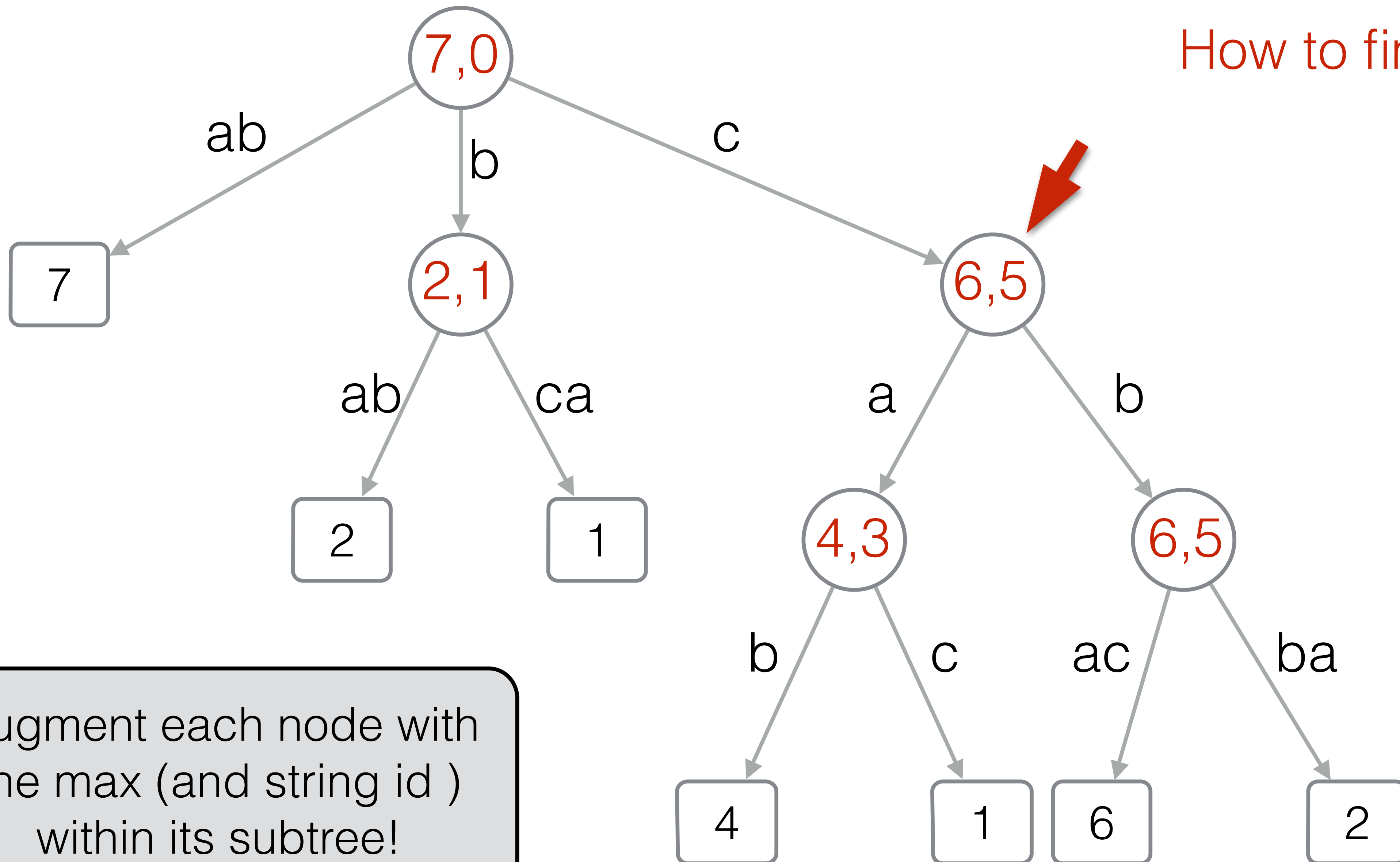
D (1), cab (4), cac (1), cbac (6), cbba (2) }

l length of strings in D

Finding Top-1

P = c

How to find Top-1?



Augment each node with the max (and string id) within its subtree!

Preprocessing time: O(n)

Extra space: O(n log n) bits

Query time: O(1)

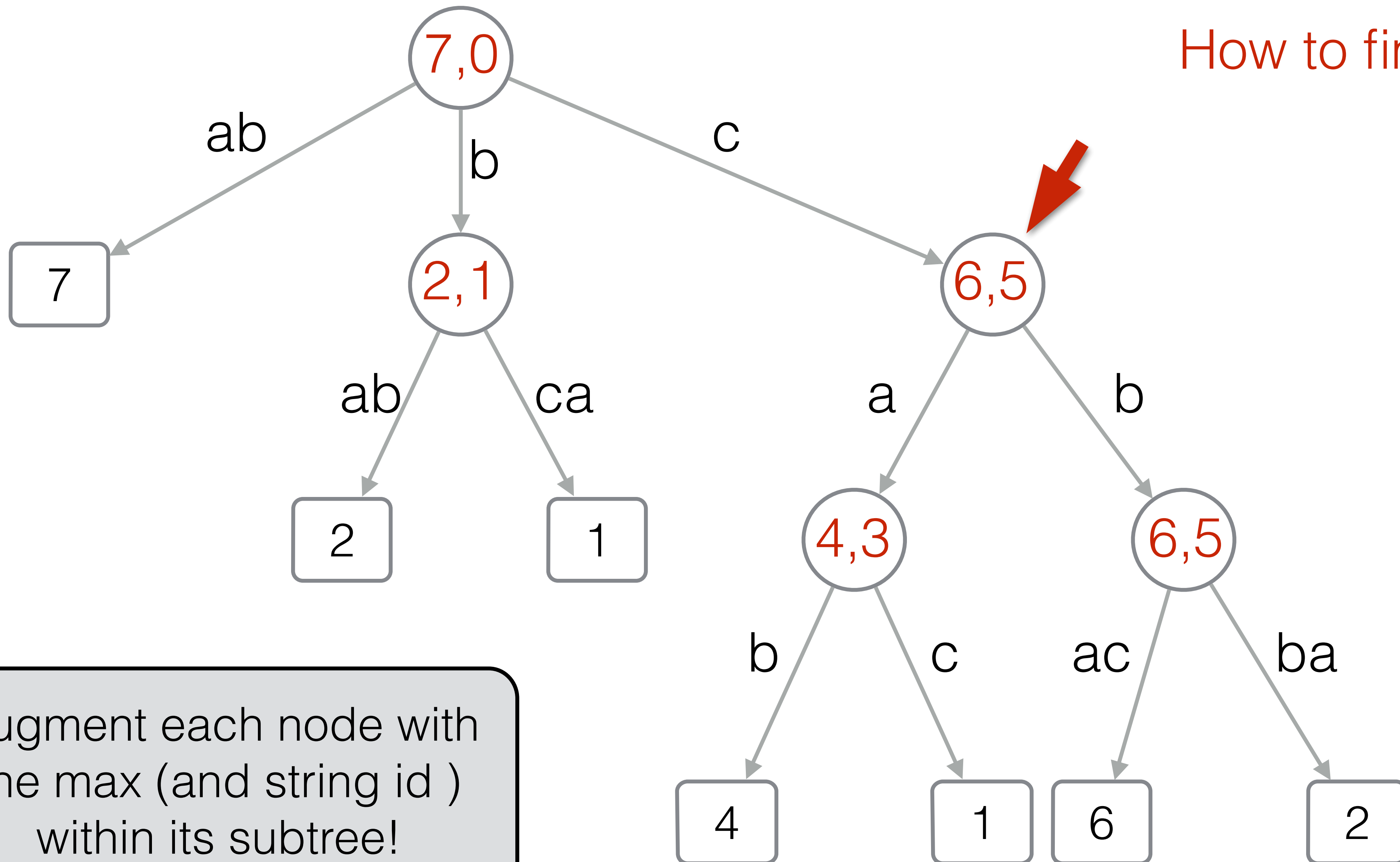
D … (1), ca … l length of strings in D

Solving Top-k?

Finding Top-1

P = c

How to find Top-1?



Augment each node with the max (and string id) within its subtree!

Preprocessing time: O(n)

Extra space: O(n log n) bits

Query time: O(1)

Solving Top-k?

- Extra space: O(k*n*log n) bits :-(
- You must know k at building time! :-(

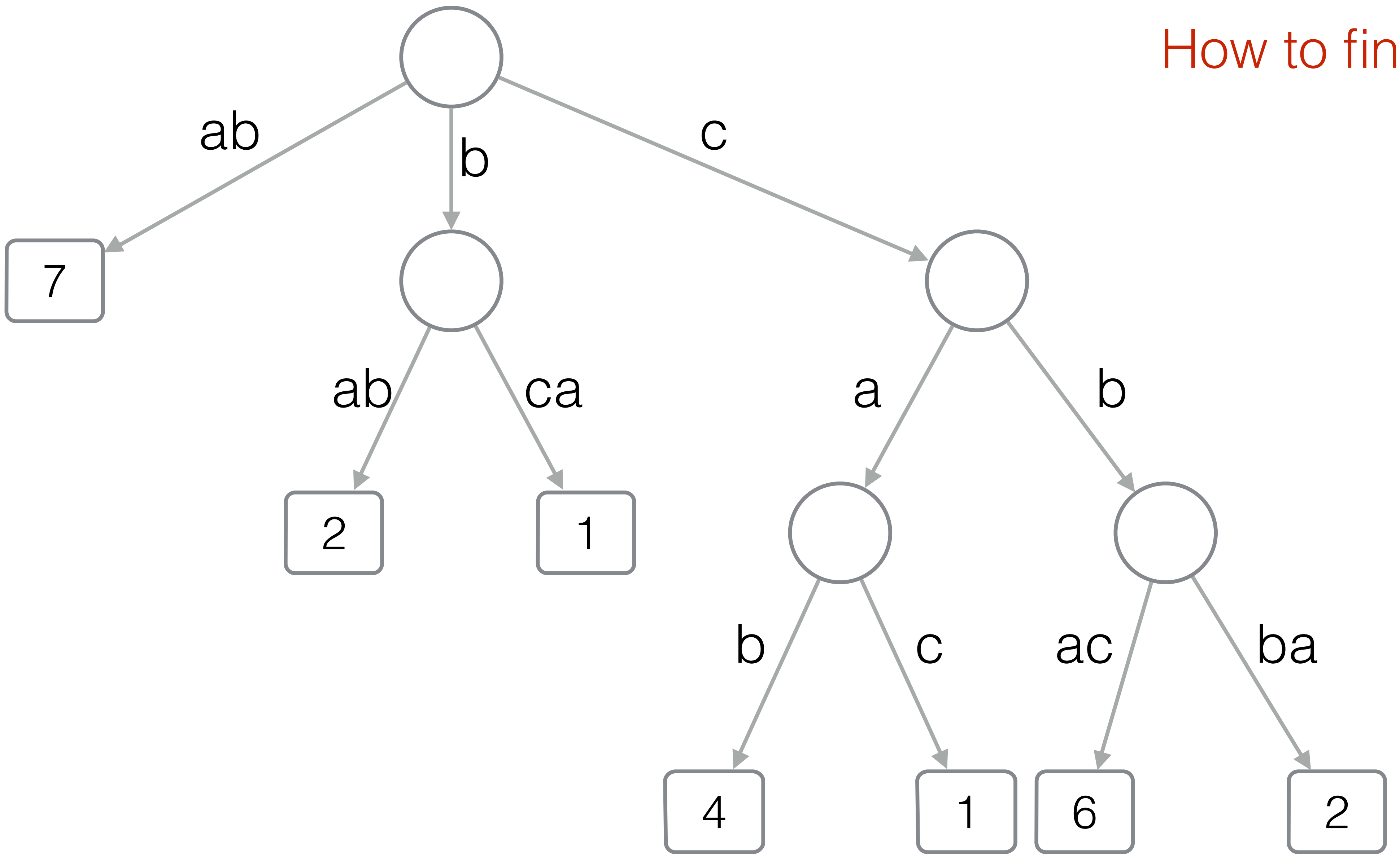
D (1), ca

l length of strings in D

Finding Top-1

P = c

How to find Top-1?



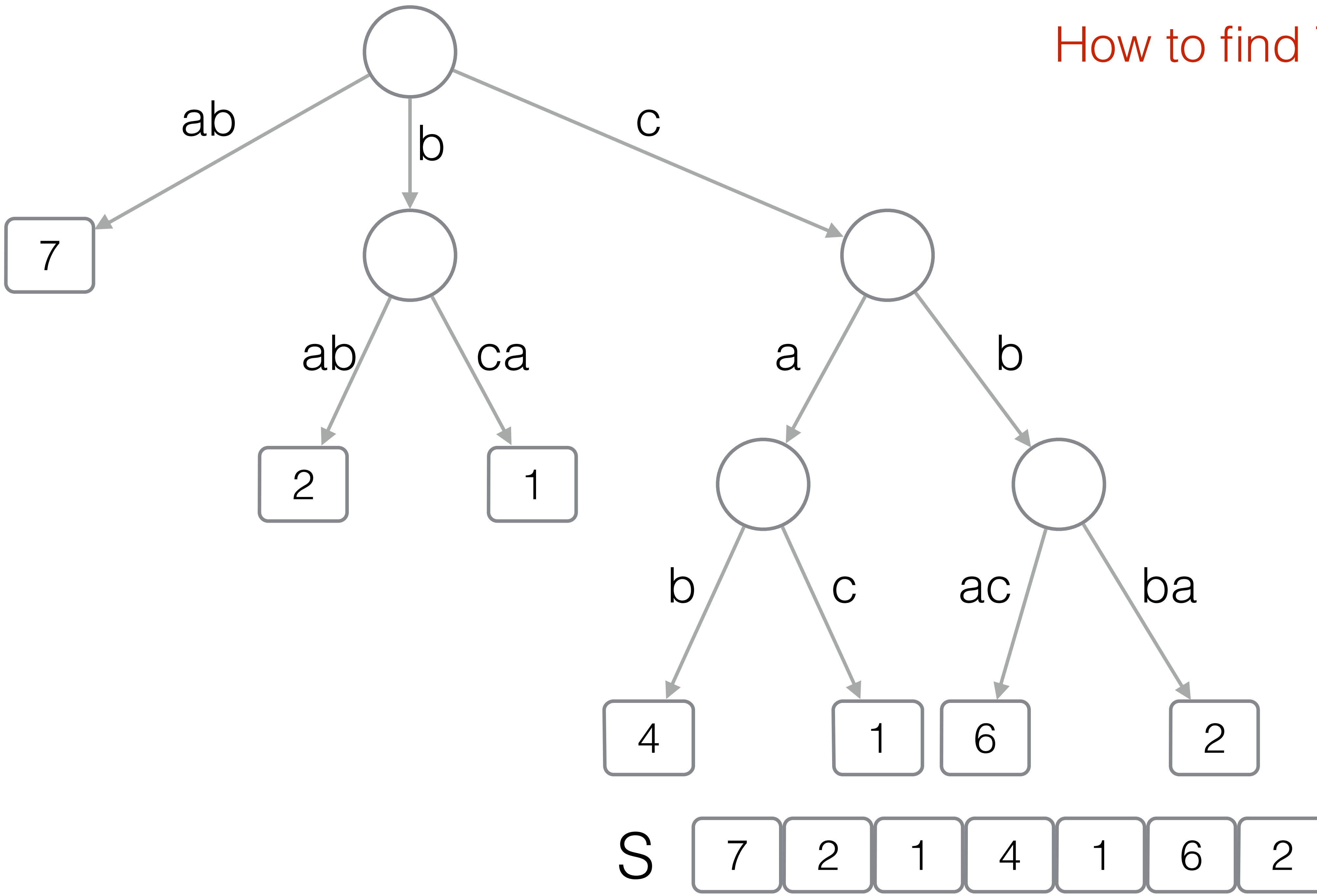
D = { ab (7), bab (2), bca (1), cab (4), cac (1), cbac (6), cbba (2) }

n = |D|, m total length of strings in D

Finding Top-1

P = c

How to find Top-1?



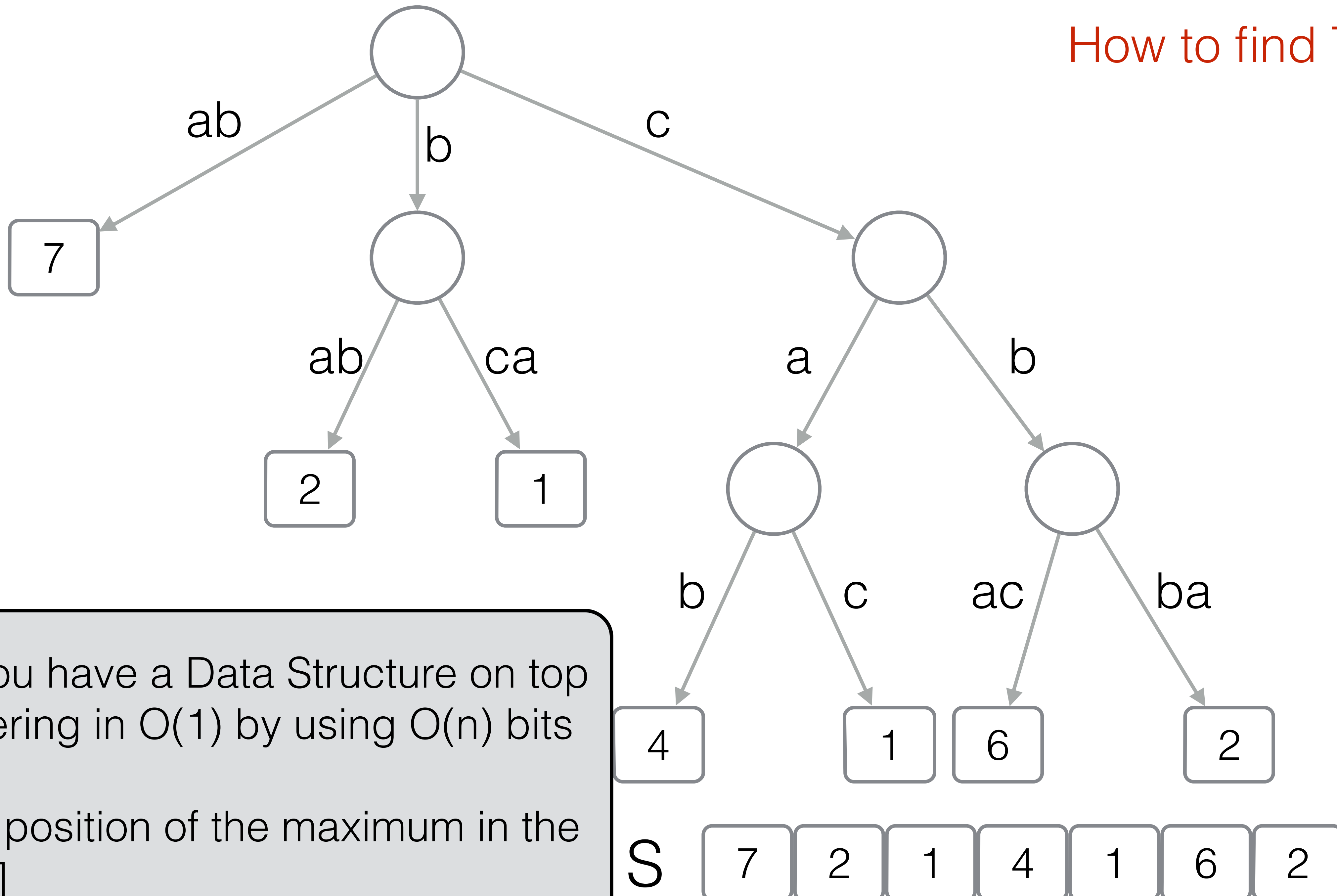
D = { ab (7), bab (2), bca (1), cab (4), cac (1), cbac (6), cbba (2) }

n = |D|, m total length of strings in D

Finding Top-1

P = c

How to find Top-1?



Assume you have a Data Structure on top of S answering in O(1) by using O(n) bits

RMQ(i,j) = position of the maximum in the range S[i,j]

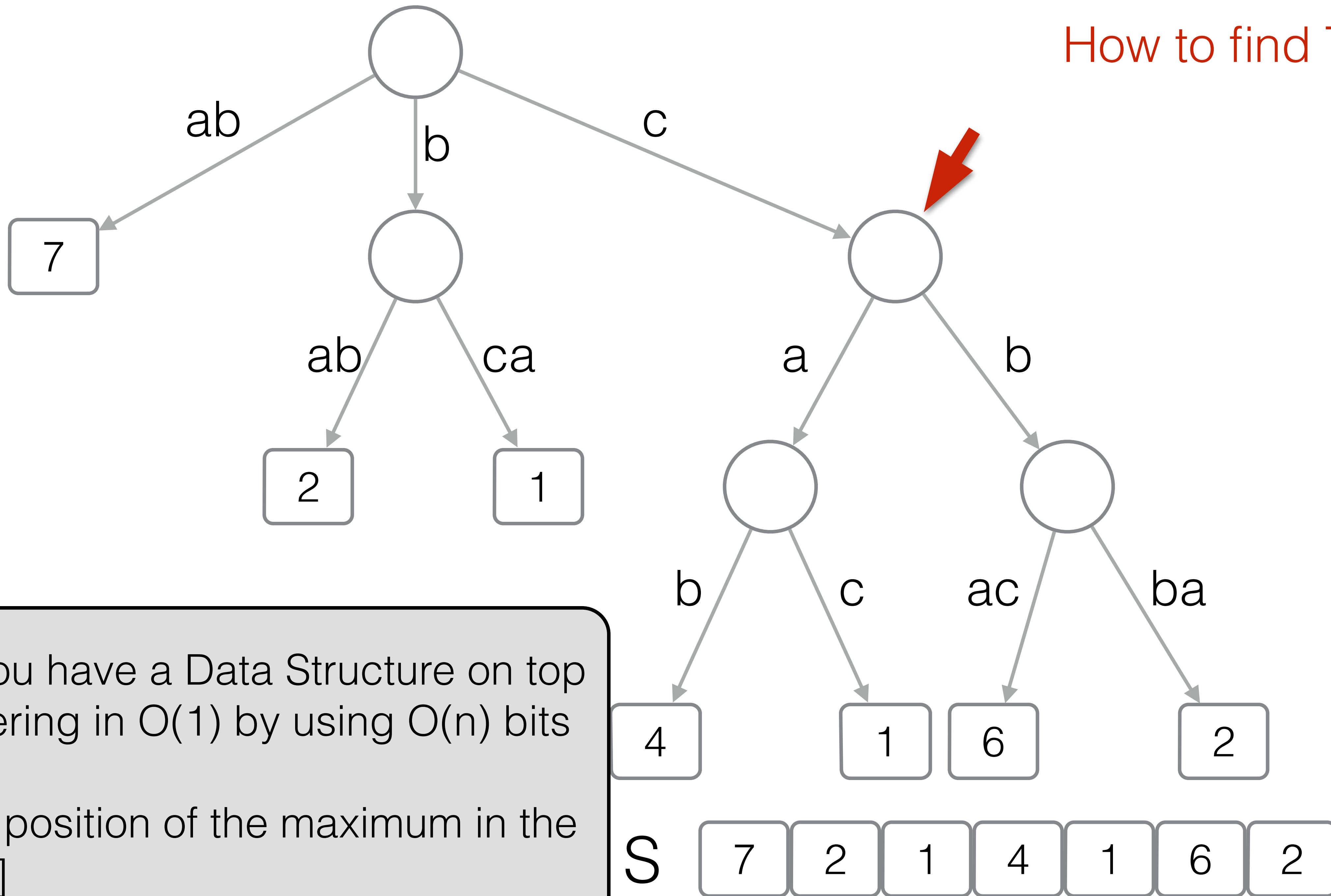
D = { ab (7), bab (2), bca (1), cab (4), cac (1), cbac (6), cbba (2) }

n = |D|, m total length of strings in D

Finding Top-1

P = c

How to find Top-1?



Assume you have a Data Structure on top of S answering in O(1) by using O(n) bits

RMQ(i,j) = position of the maximum in the range S[i,j]

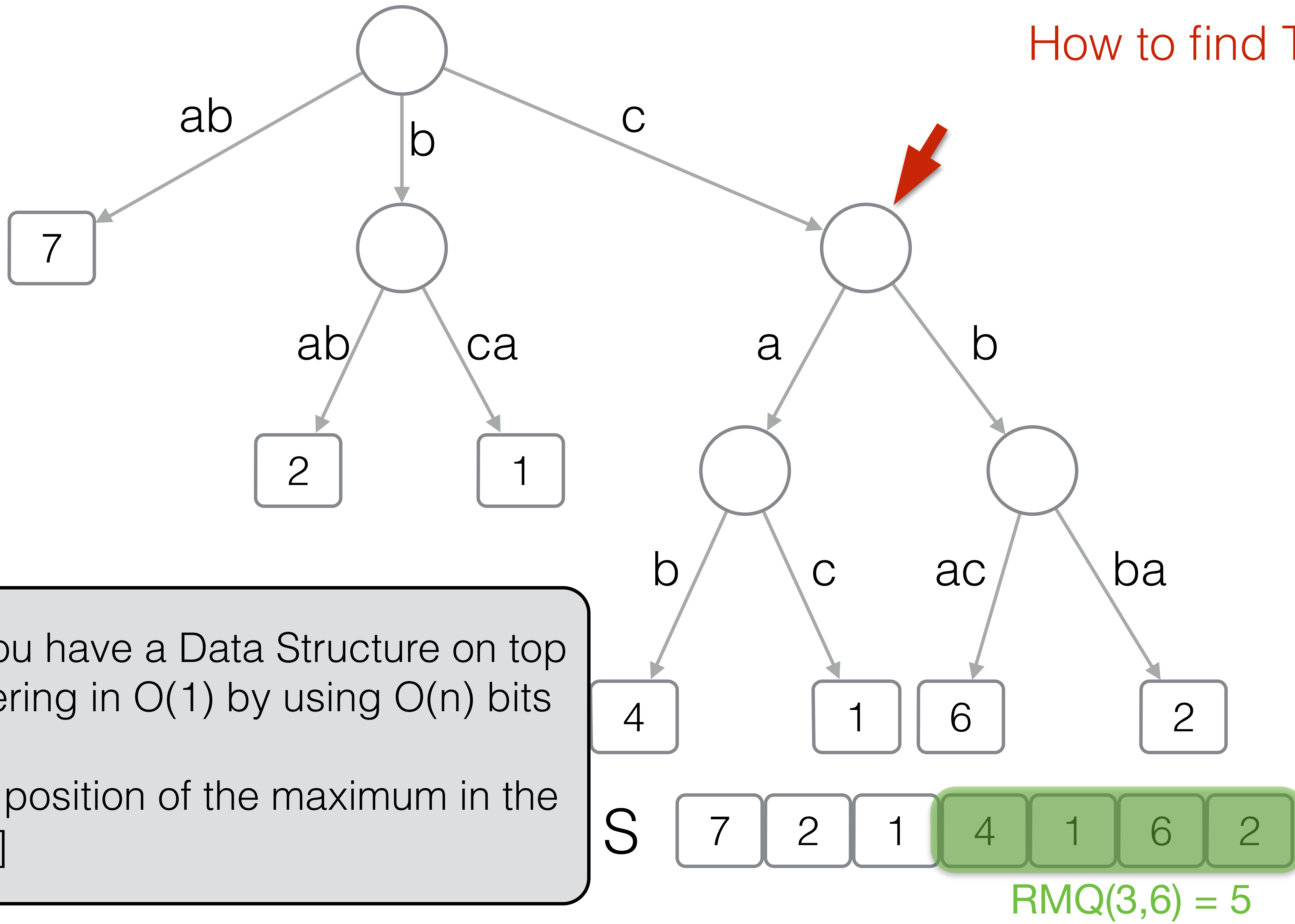
D = { ab (7), bab (2), bca (1), cab (4), cac (1), cbac (6), cbba (2) }

n = |D|, m total length of strings in D

Finding Top-1

P = c

How to find Top-1?



Assume you have a Data Structure on top of S answering in O(1) by using O(n) bits

RMQ(i,j) = position of the maximum in the range S[i,j]

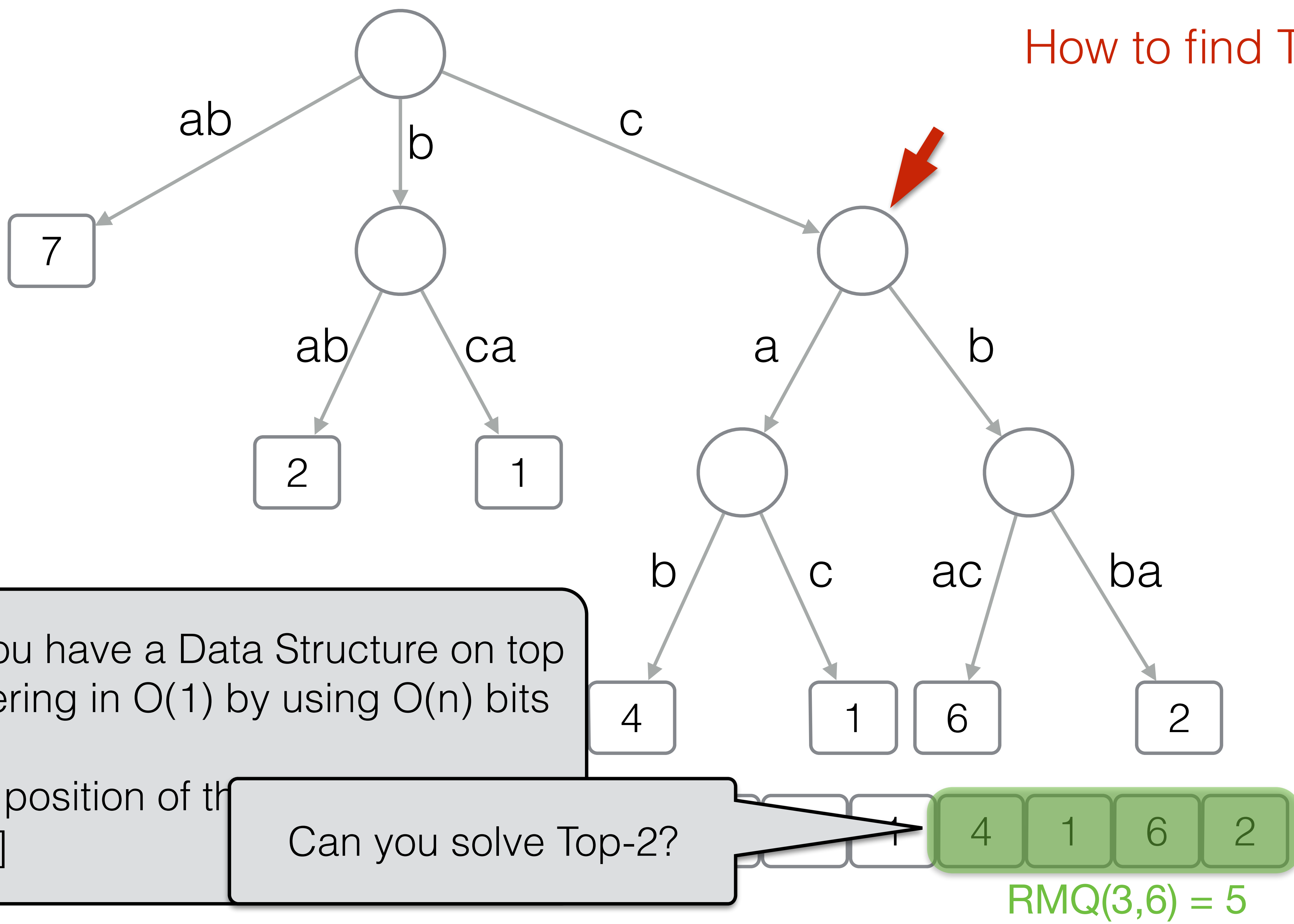
D = { ab (7), bab (2), bca (1), cab (4), cac (1), cbac (6), cbba (2) }

n = |D|, m total length of strings in D

Finding Top-1

P = c

How to find Top-1?



Assume you have a Data Structure on top of S answering in O(1) by using O(n) bits

RMQ(i,j) = position of th
range S[i,j]

Can you solve Top-2?

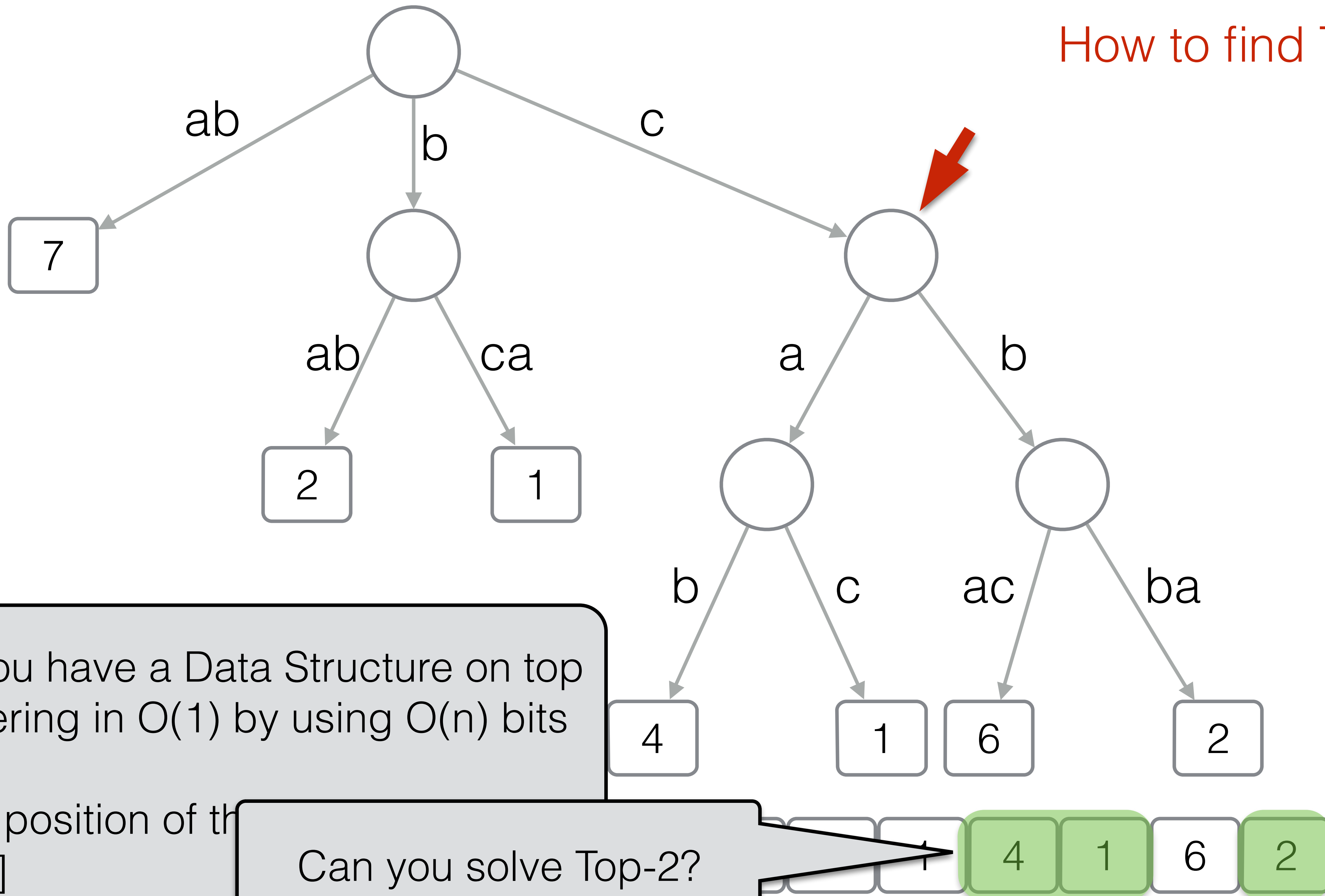
D = { ab (7), bab (2), bca (1), cab (4), cac (1), cbac (6), cbba (2) }

n = |D|, m total length of strings in D

Finding Top-1

P = c

How to find Top-1?



Assume you have a Data Structure on top of S answering in O(1) by using O(n) bits

RMQ(i,j) = position of th
range S[i,j]

Can you solve Top-2?

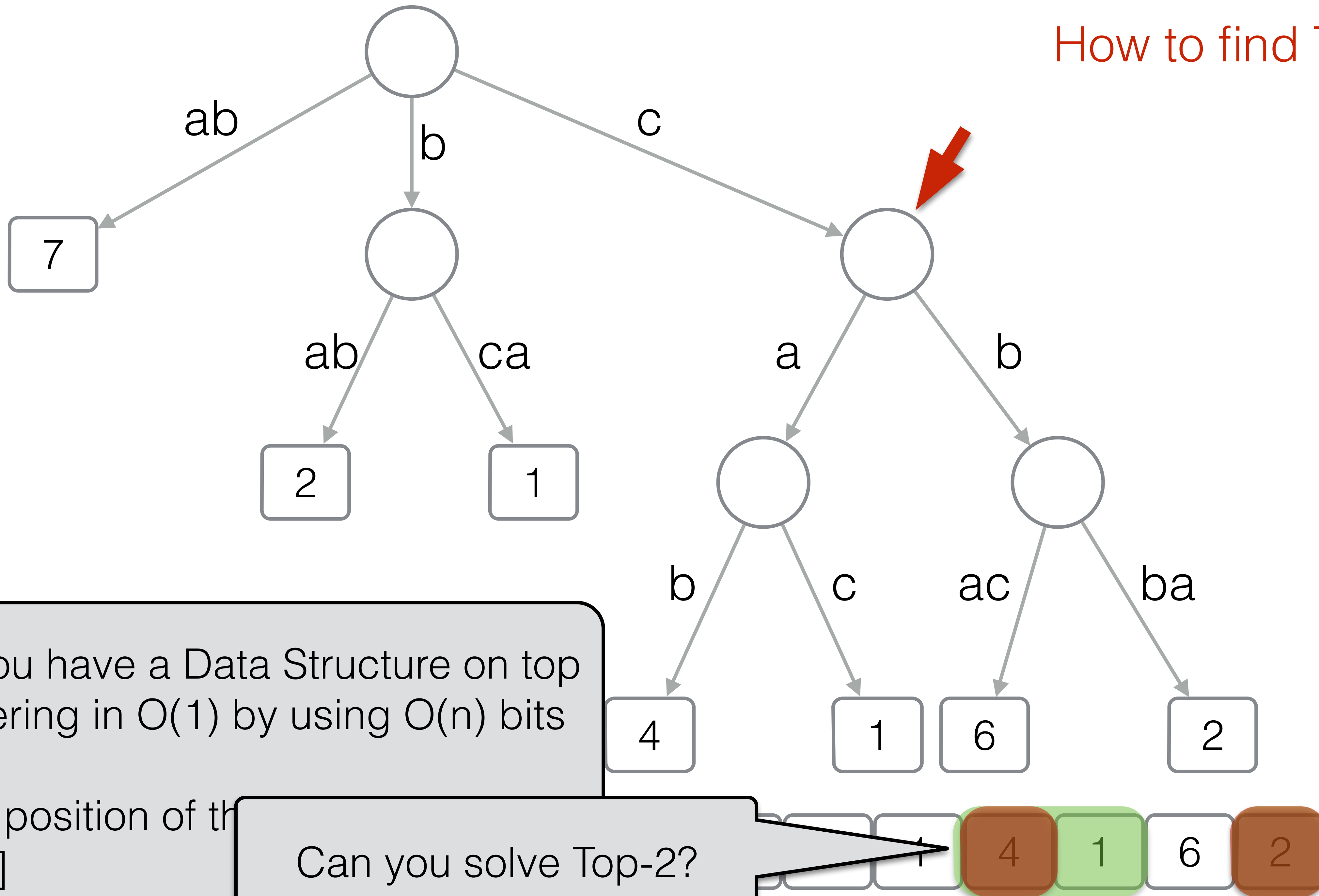
D = { ab (7), bab (2), bca (1), cab (4), cac (1), cbac (6), cbba (2) }

n = |D|, m total length of strings in D

Finding Top-1

P = c

How to find Top-1?



Assume you have a Data Structure on top of S answering in O(1) by using O(n) bits

RMQ(i,j) = position of th
range S[i,j]

Can you solve Top-2?

D = { ab (7), bab (2), bca (1), cab (4), cac (1), cbac (6), cbba (2) }

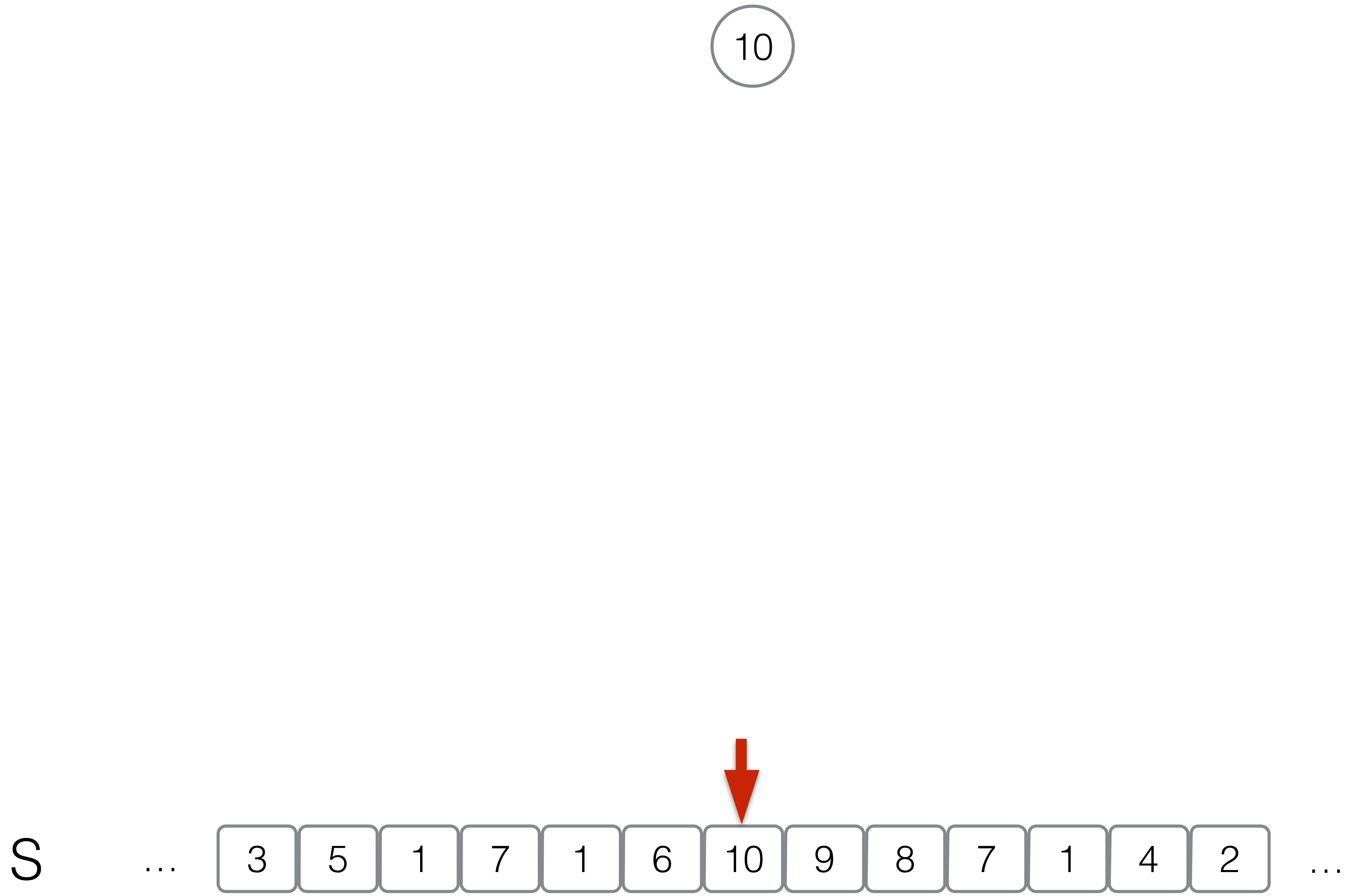
n = |D|, m total length of strings in D

Finding Top-k

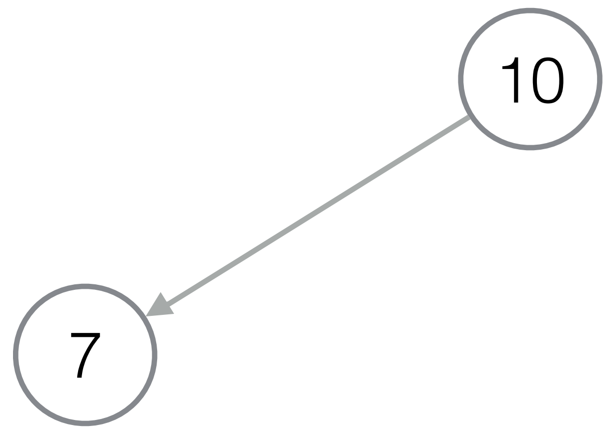
Finding Top-k

S … 3 5 1 7 1 6 10 9 8 7 1 4 2 …

Finding Top-k

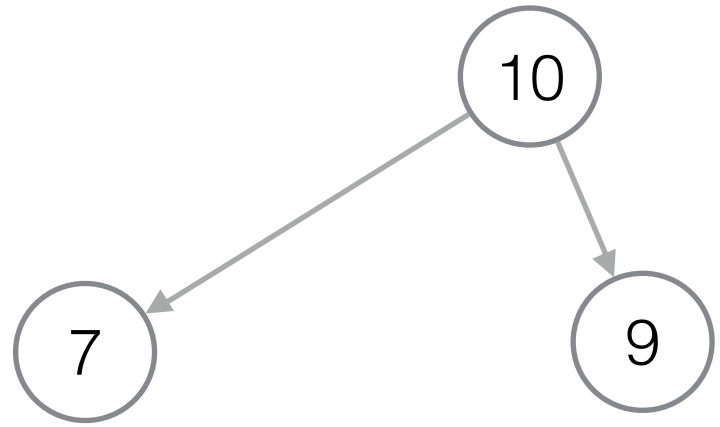


Finding Top-k



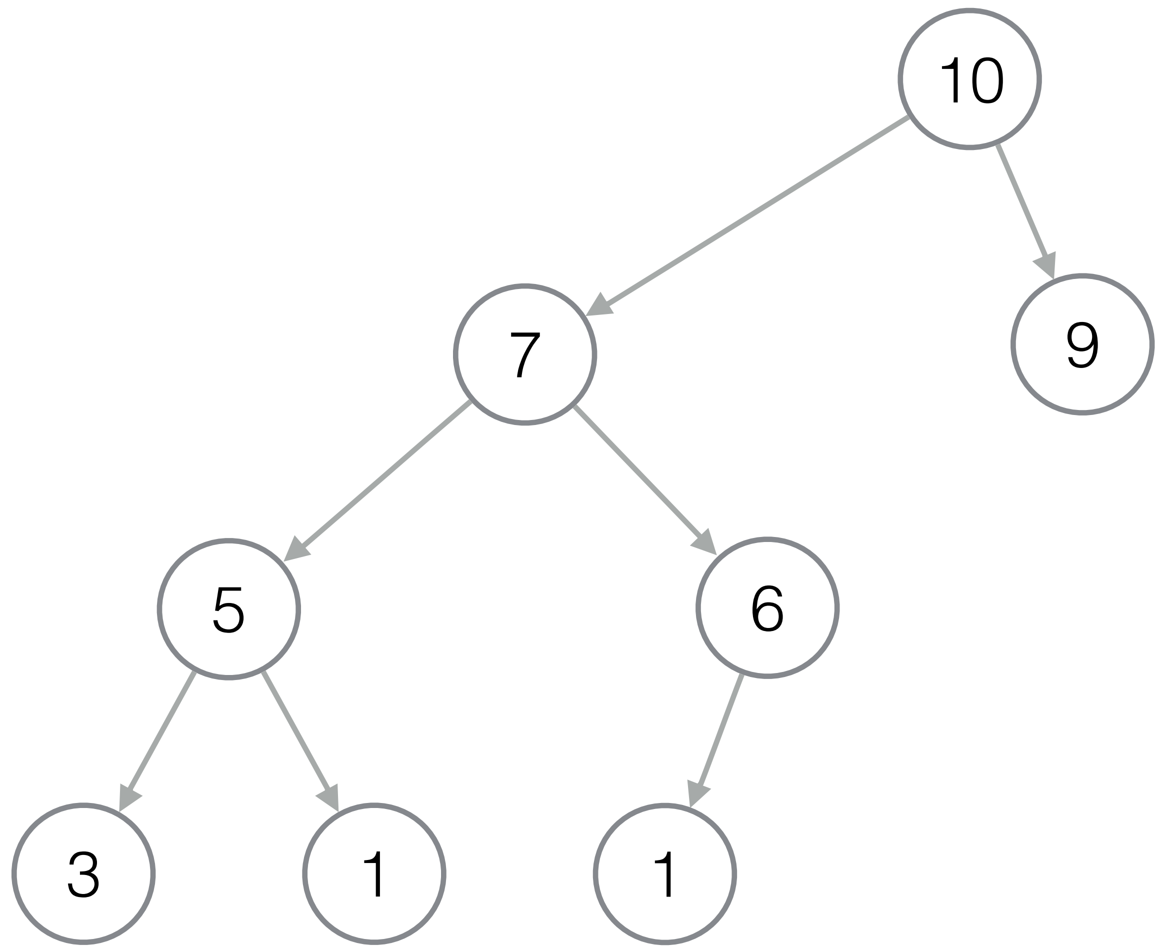
S	…	3	5	1	7	1	6	10	9	8	7	1	4	2	…

Finding Top-k



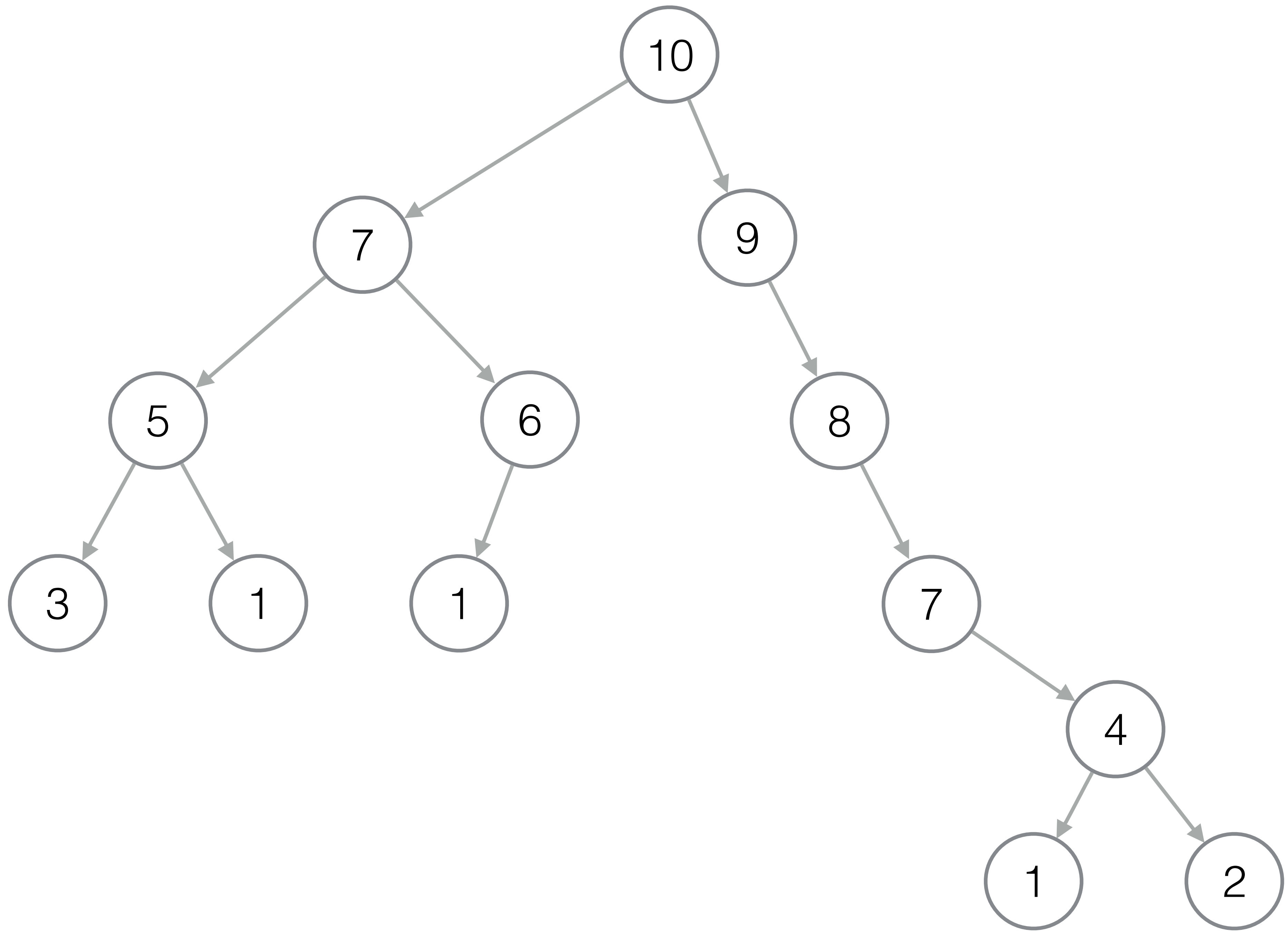
S … 3 5 1 7 1 6 10 9 8 7 1 4 2 …

Finding Top-k



S	…	3	5	1	7	1	6	10	9	8	7	1	4	2	…

Finding Top-k

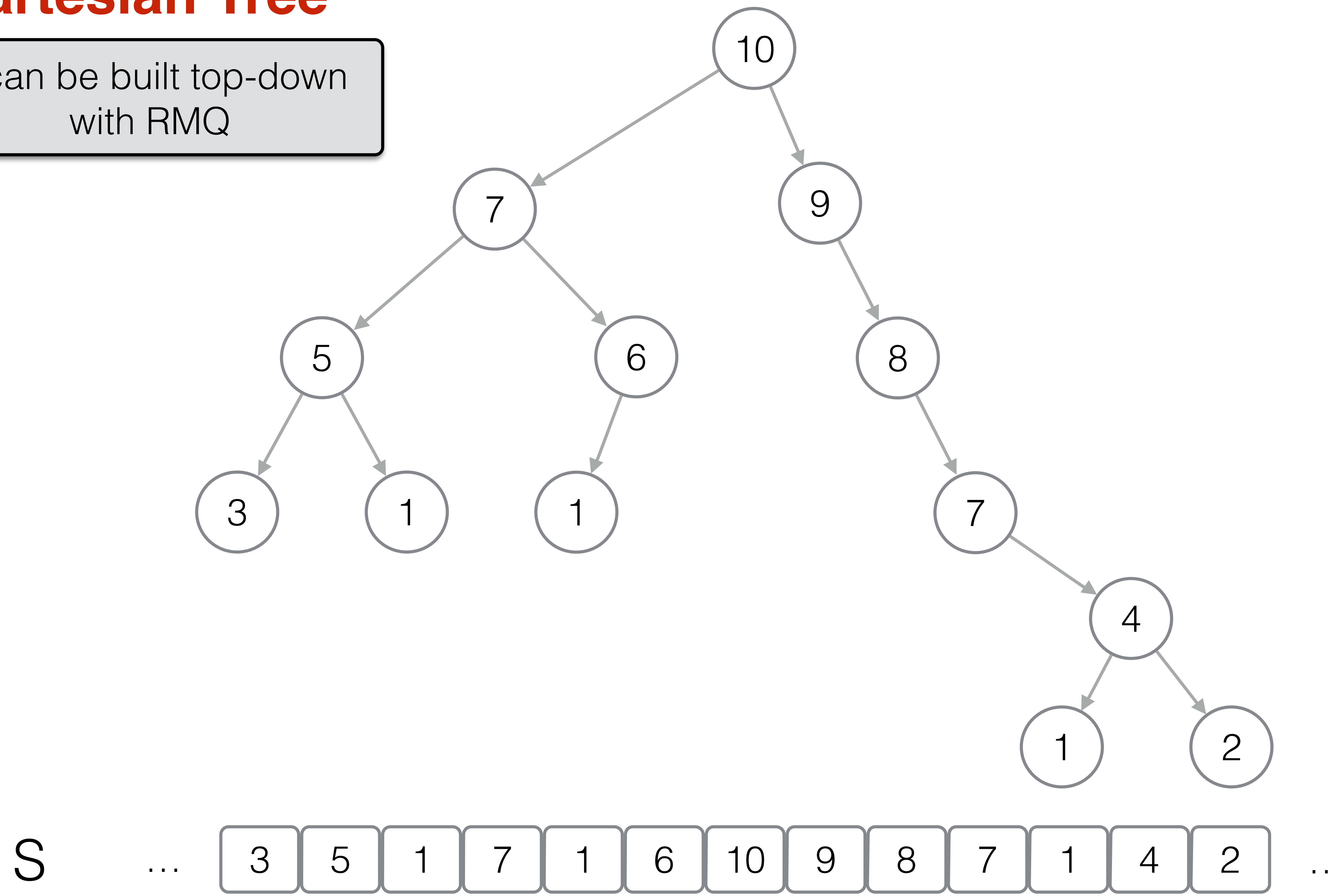


S	…	3	5	1	7	1	6	10	9	8	7	1	4	2	…

Finding Top-k

Cartesian Tree

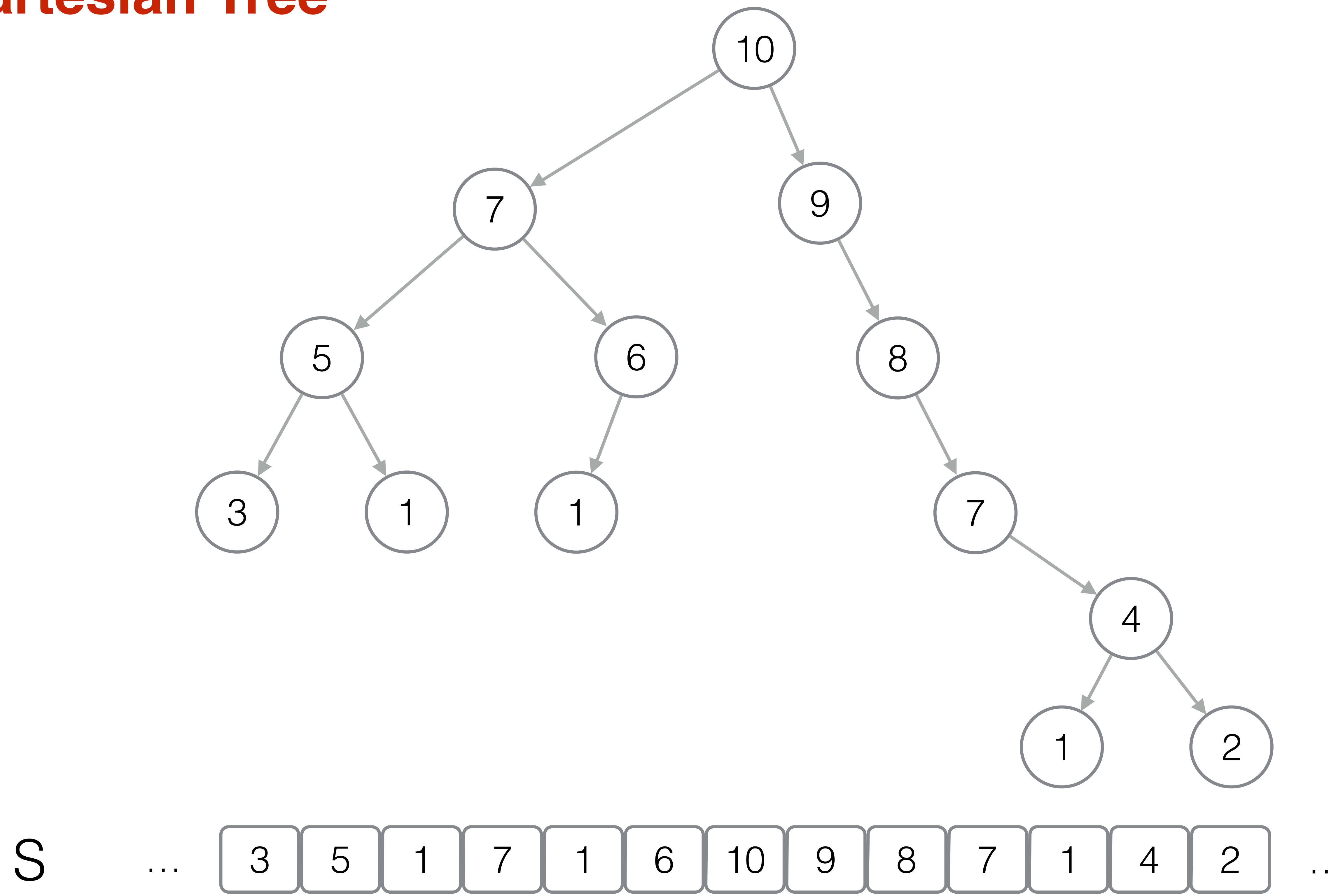
It can be built top-down
with RMQ



Finding Top-k

How to find Top-k?

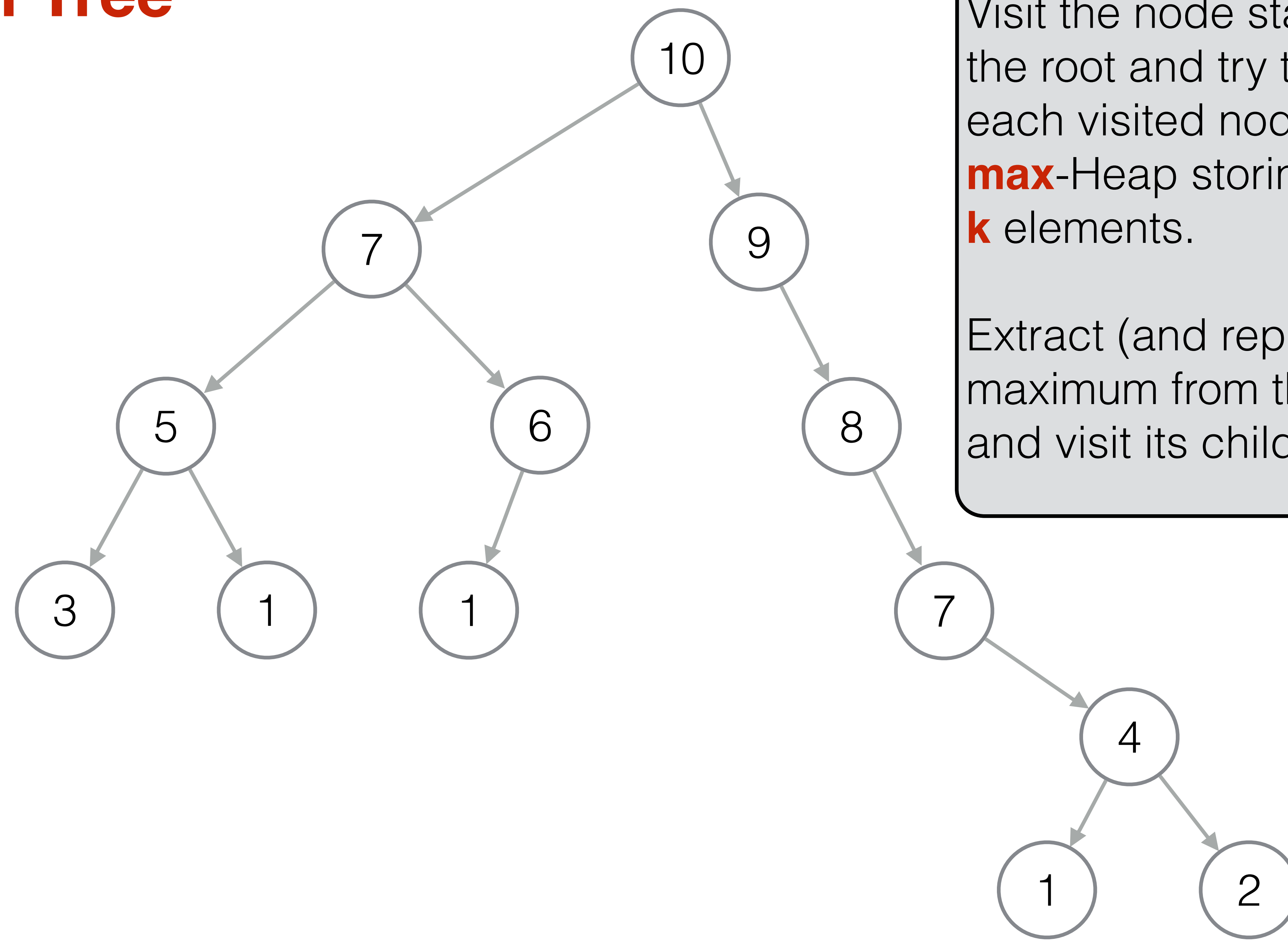
Cartesian Tree



Finding Top-k

How to find Top-k?

Cartesian Tree



S	…	3	5	1	7	1	6	10	9	8	7	1	4	2	…

Visit the node starting from the root and try to insert each visited node in a **max**-Heap storing at most **k** elements.

Extract (and report) the maximum from the heap and visit its children.

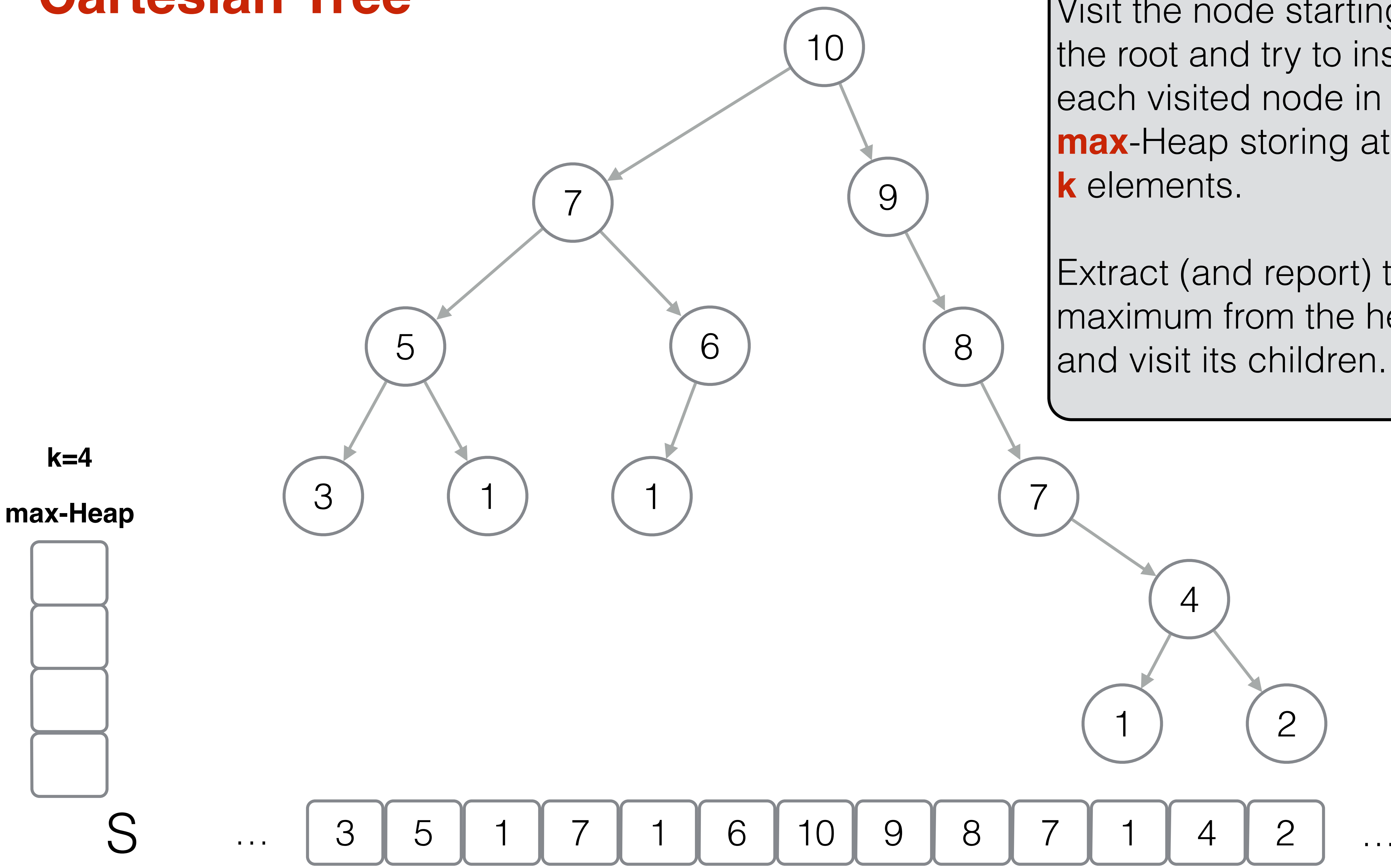
Finding Top-k

How to find Top-k?

Visit the node starting from the root and try to insert each visited node in a **max**-Heap storing at most **k** elements.

Extract (and report) the maximum from the heap and visit its children.

Cartesian Tree



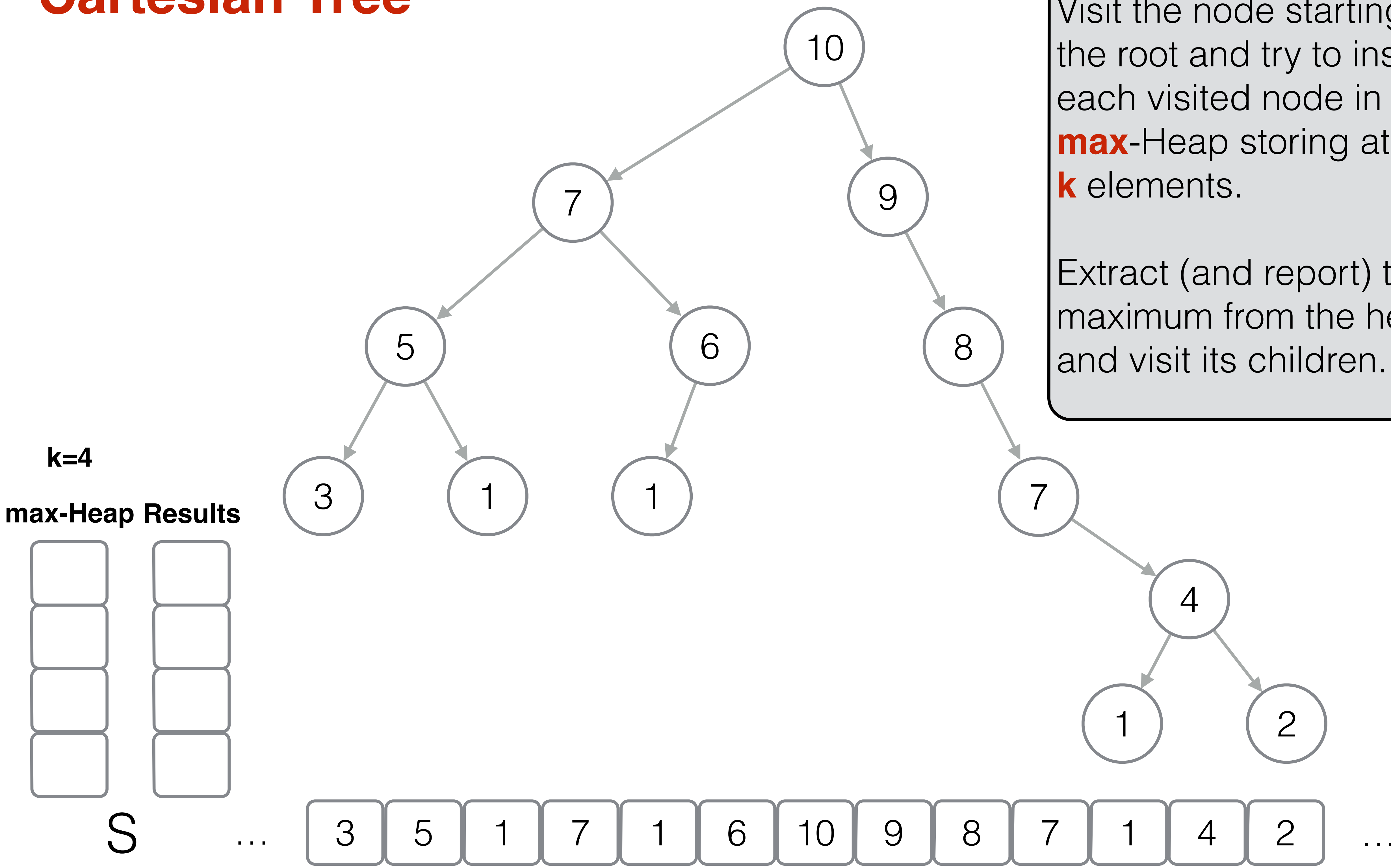
Finding Top-k

How to find Top-k?

Visit the node starting from the root and try to insert each visited node in a **max**-Heap storing at most **k** elements.

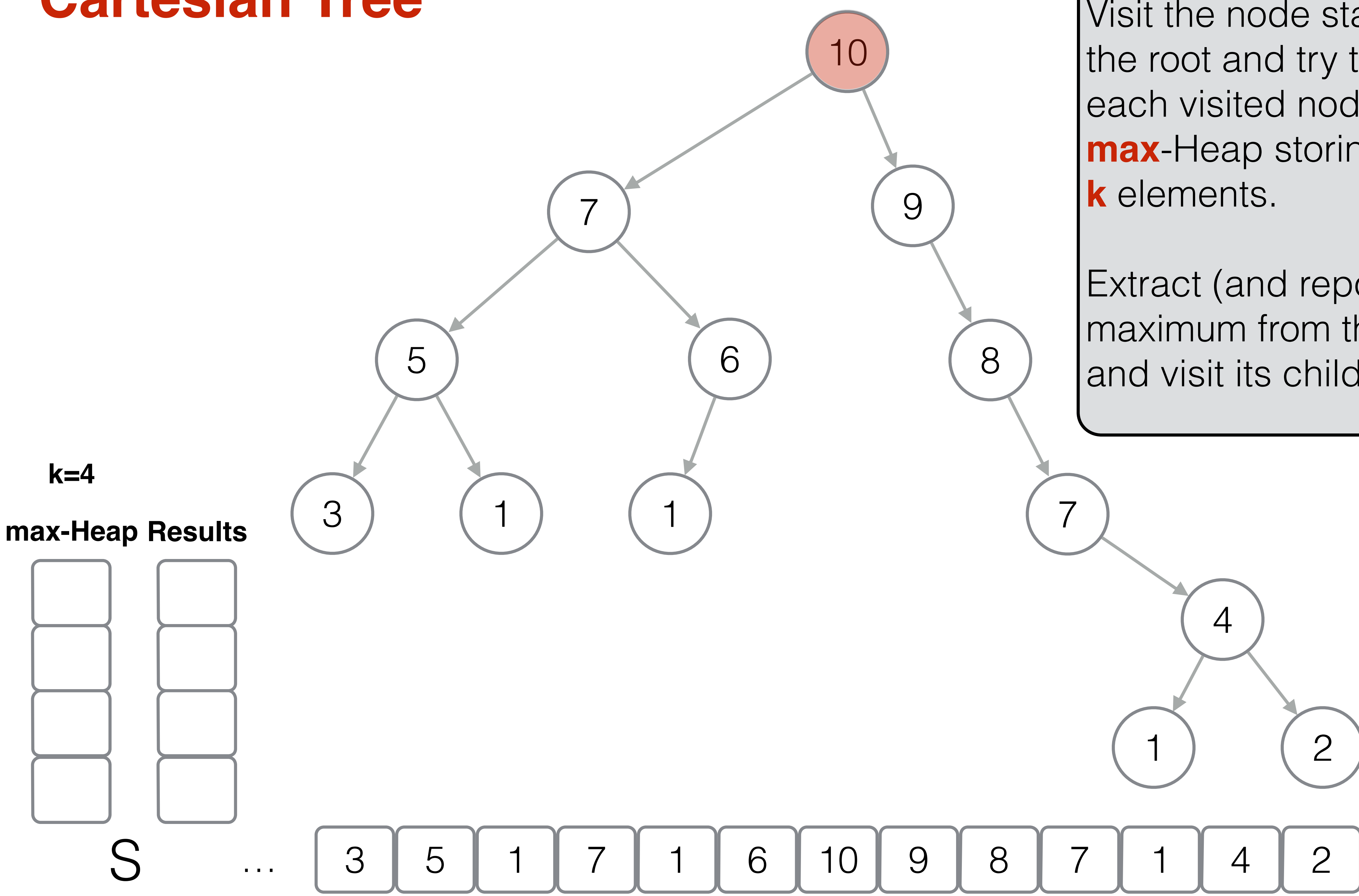
Extract (and report) the maximum from the heap and visit its children.

Cartesian Tree



Finding Top-k

Cartesian Tree



How to find Top-k?

Visit the node starting from the root and try to insert each visited node in a **max**-Heap storing at most **k** elements.

Extract (and report) the maximum from the heap and visit its children.

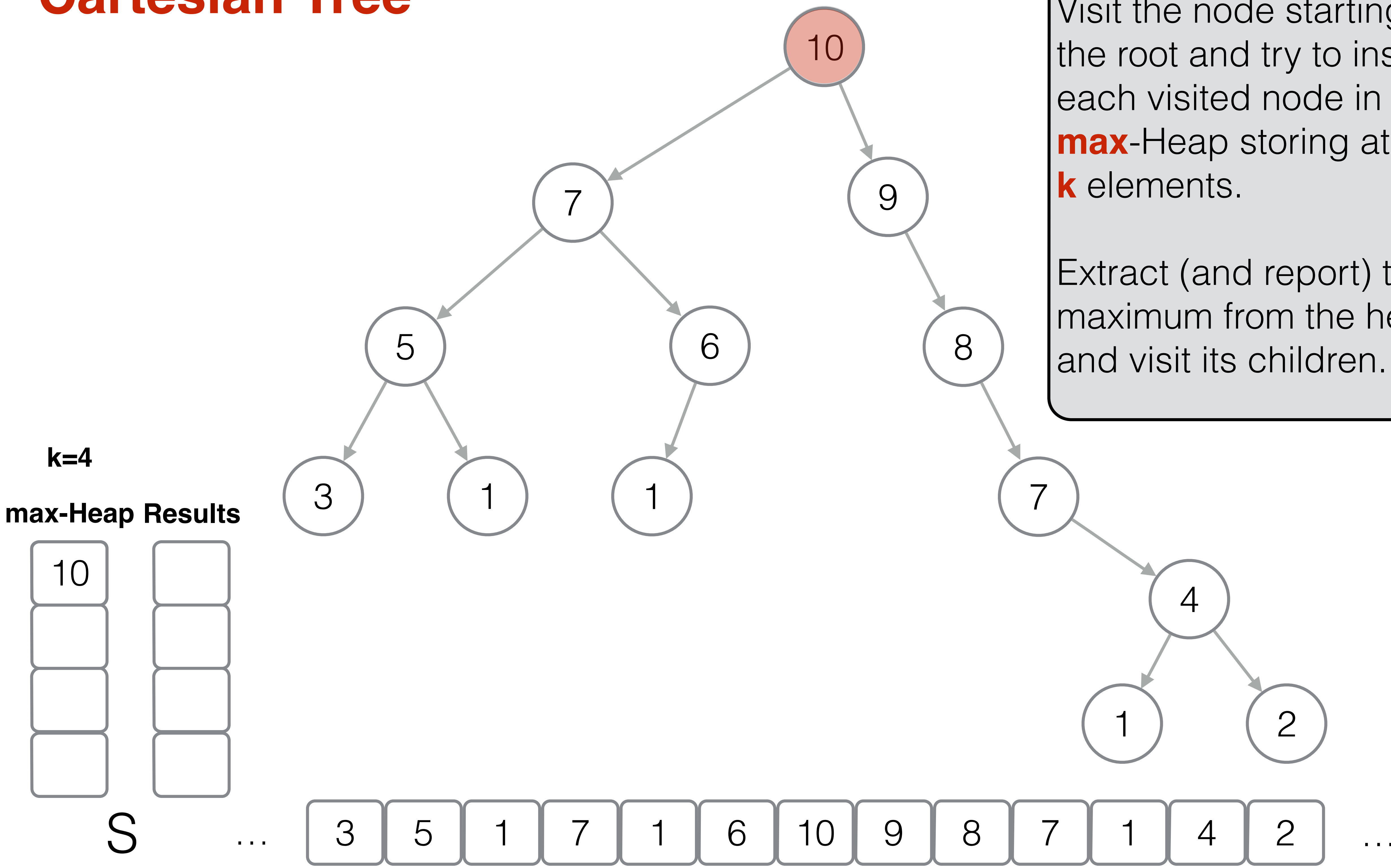
Finding Top-k

How to find Top-k?

Cartesian Tree

Visit the node starting from the root and try to insert each visited node in a **max**-Heap storing at most **k** elements.

Extract (and report) the maximum from the heap and visit its children.



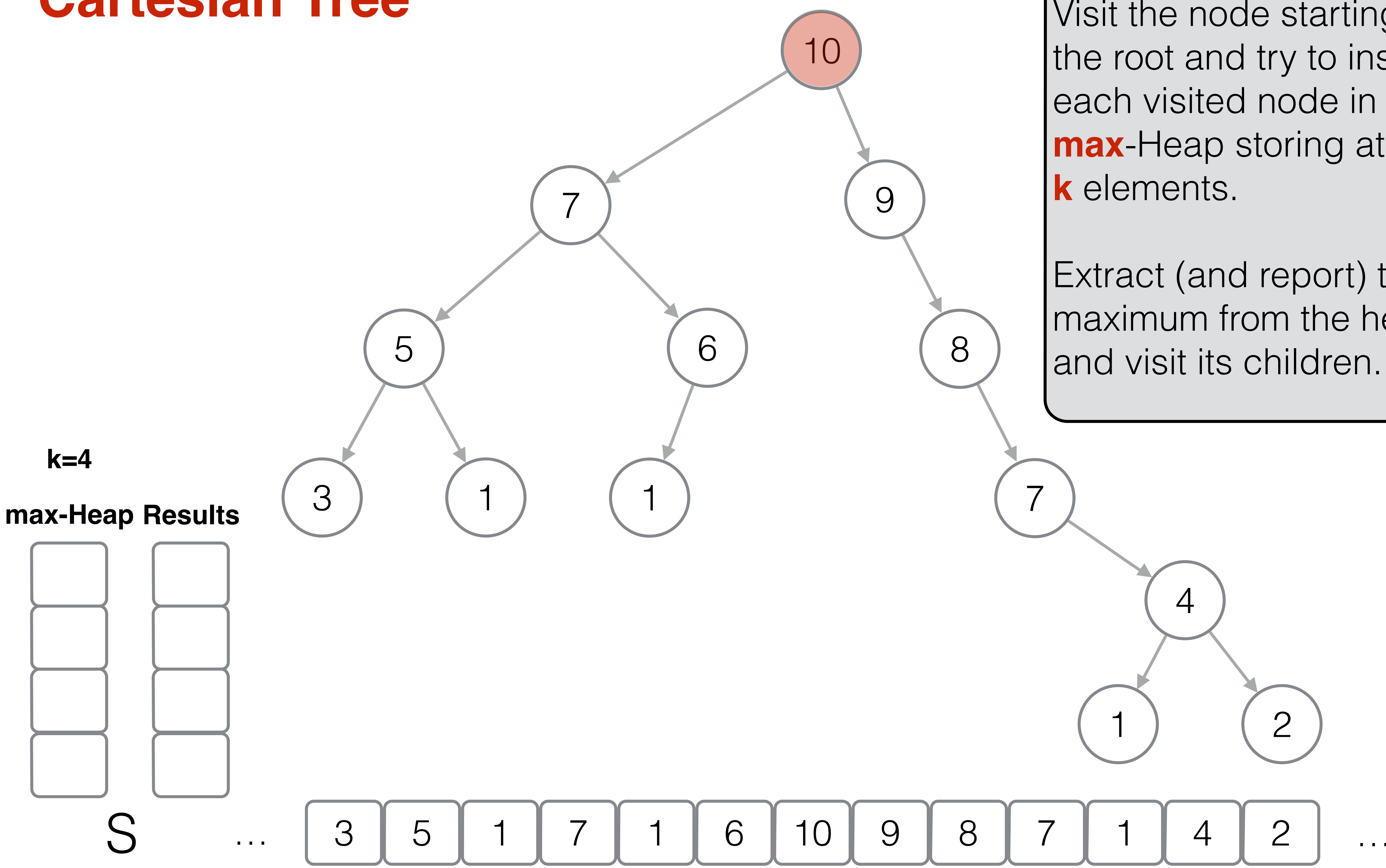
Finding Top-k

How to find Top-k?

Visit the node starting from the root and try to insert each visited node in a **max**-Heap storing at most **k** elements.

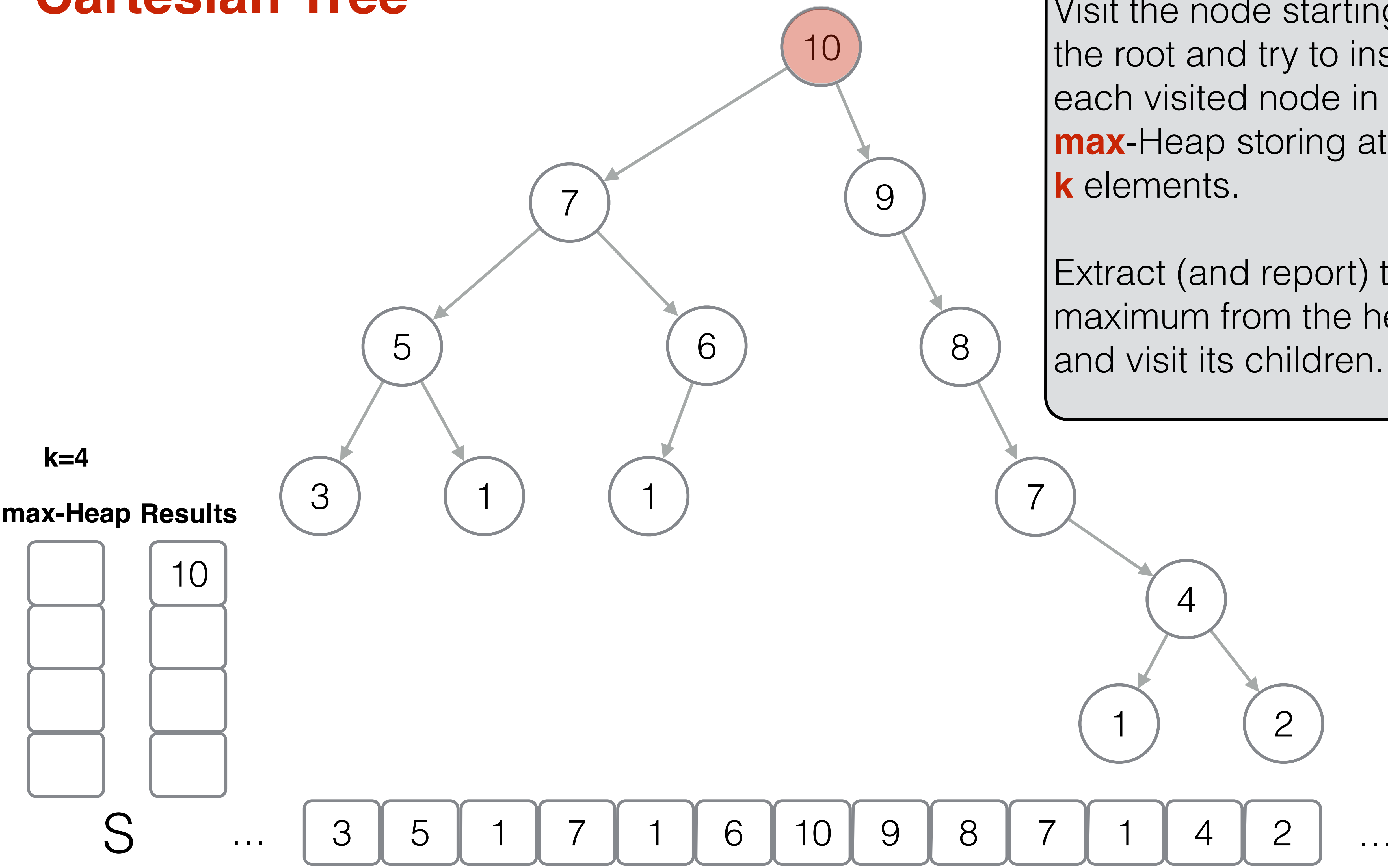
Extract (and report) the maximum from the heap and visit its children.

Cartesian Tree



Finding Top-k

Cartesian Tree



How to find Top-k?

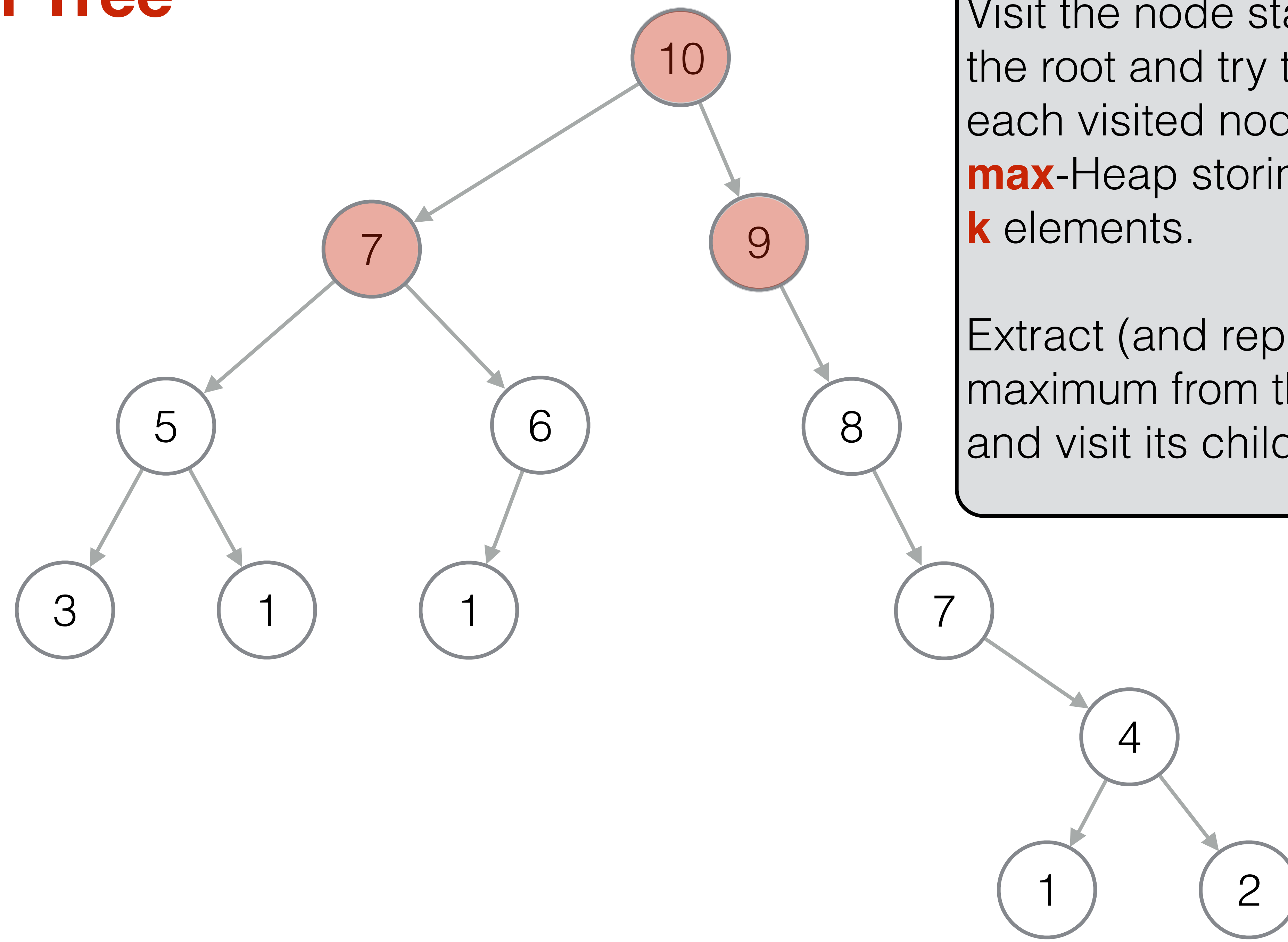
Visit the node starting from the root and try to insert each visited node in a **max**-Heap storing at most **k** elements.

Extract (and report) the maximum from the heap and visit its children.

Finding Top-k

How to find Top-k?

Cartesian Tree



Visit the node starting from the root and try to insert each visited node in a **max**-Heap storing at most **k** elements.

Extract (and report) the maximum from the heap and visit its children.

k=4

max-Heap	**Results**
	10

S … | 3 | 5 | 1 | 7 | 1 | 6 | 10 | 9 | 8 | 7 | 1 | 4 | 2 | …

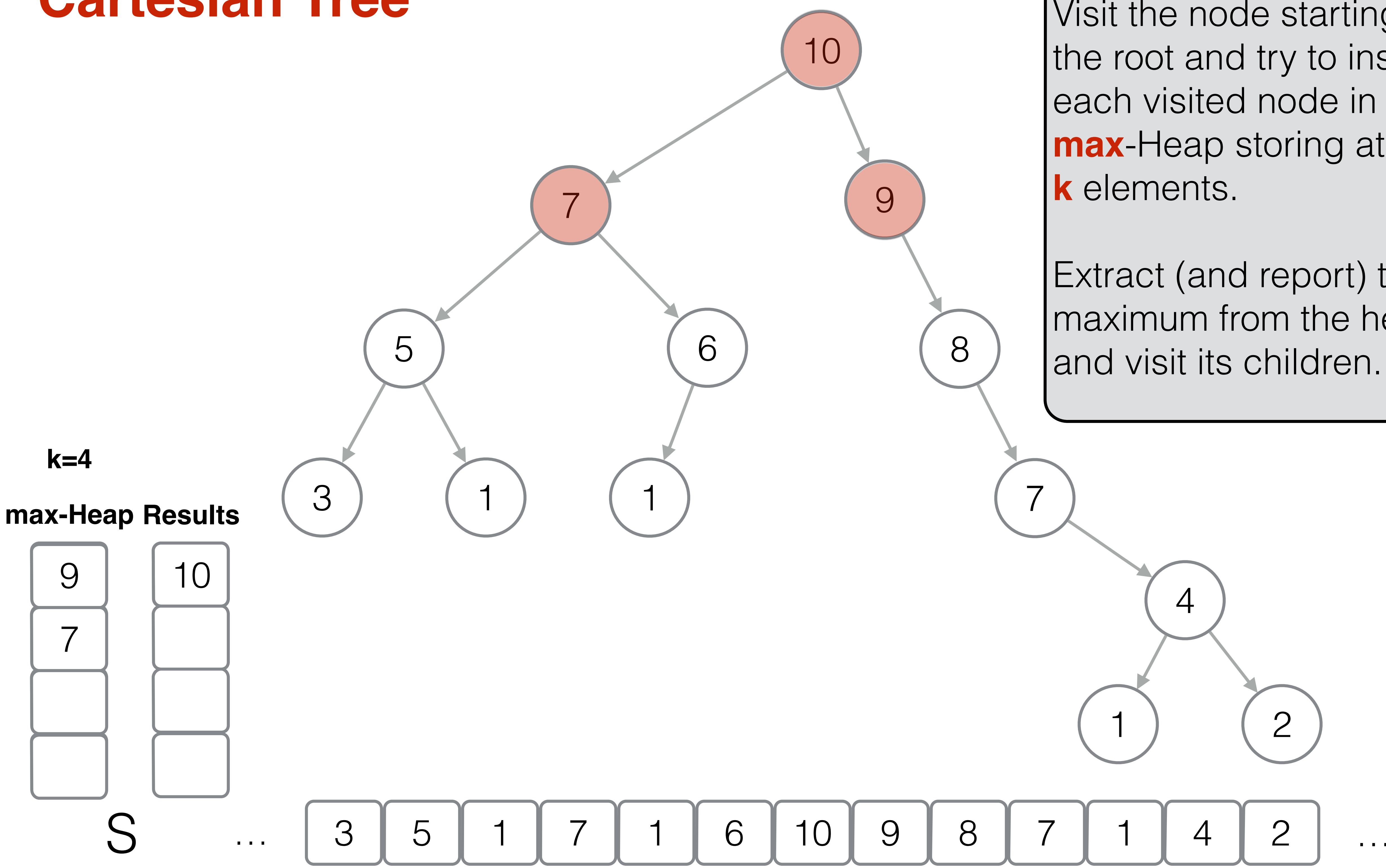
Finding Top-k

Cartesian Tree

How to find Top-k?

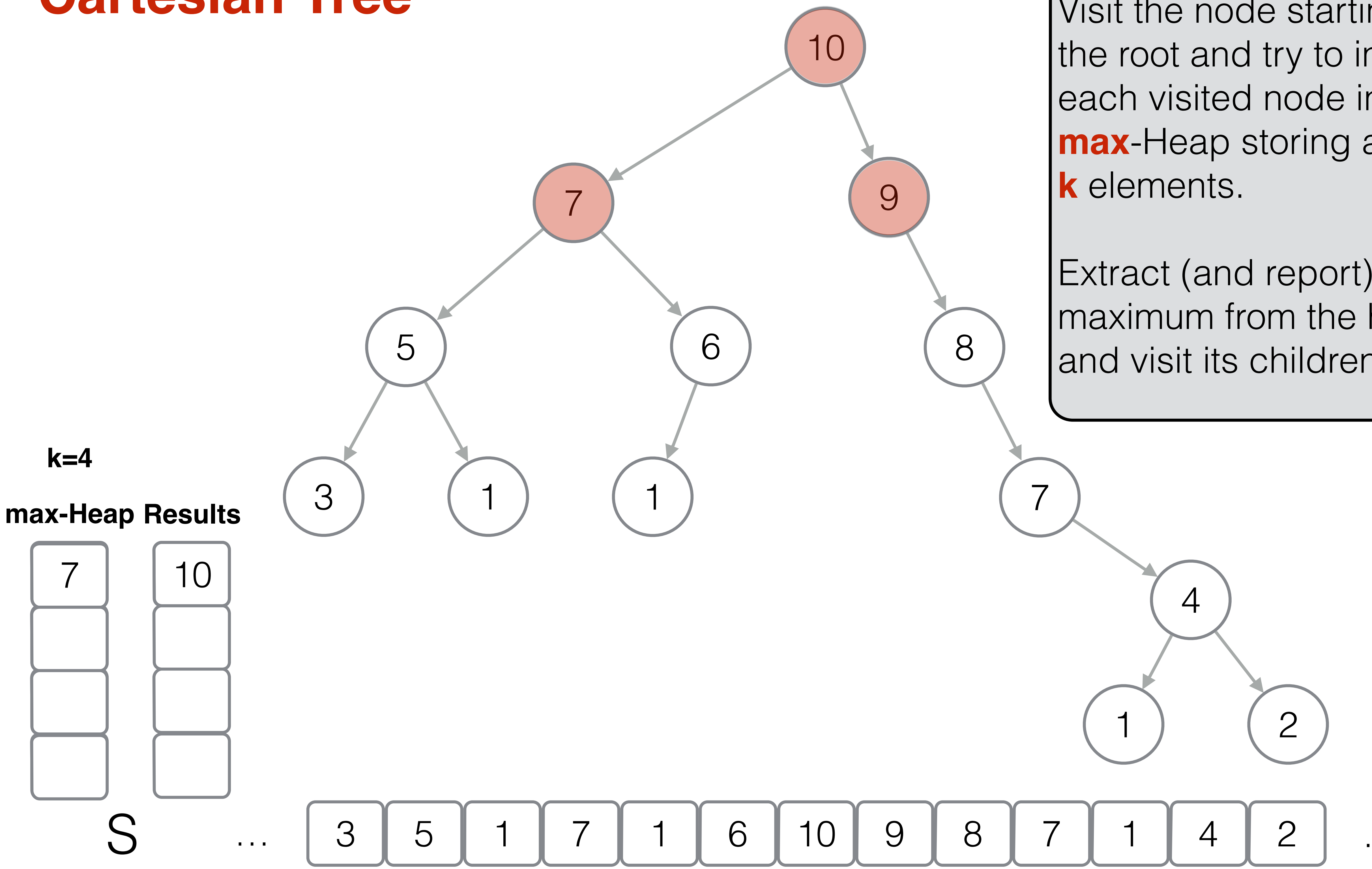
Visit the node starting from the root and try to insert each visited node in a **max**-Heap storing at most **k** elements.

Extract (and report) the maximum from the heap and visit its children.



Finding Top-k

Cartesian Tree



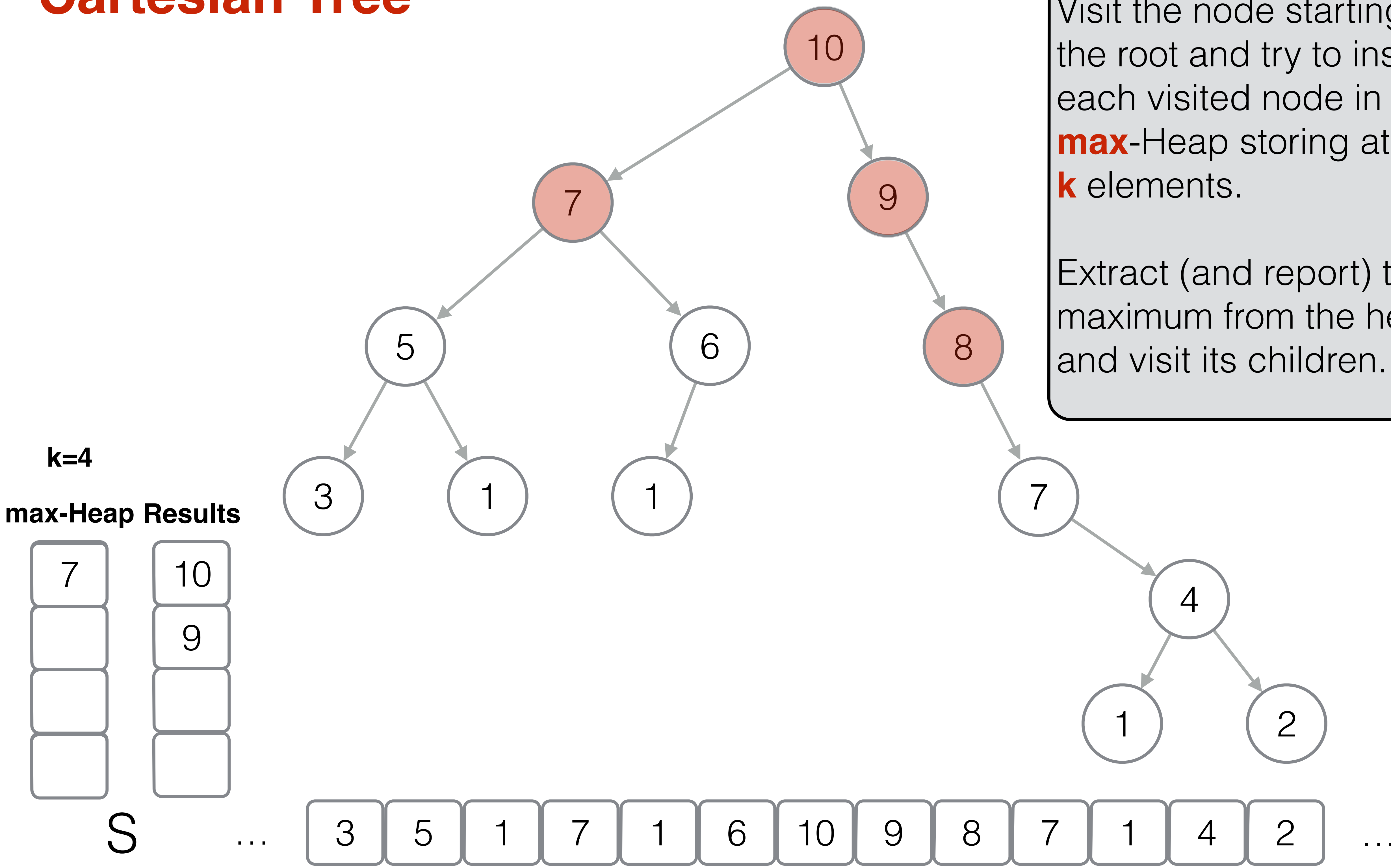
How to find Top-k?

Visit the node starting from the root and try to insert each visited node in a **max**-Heap storing at most **k** elements.

Extract (and report) the maximum from the heap and visit its children.

Finding Top-k

Cartesian Tree



How to find Top-k?

Visit the node starting from the root and try to insert each visited node in a **max**-Heap storing at most **k** elements.

Extract (and report) the maximum from the heap and visit its children.

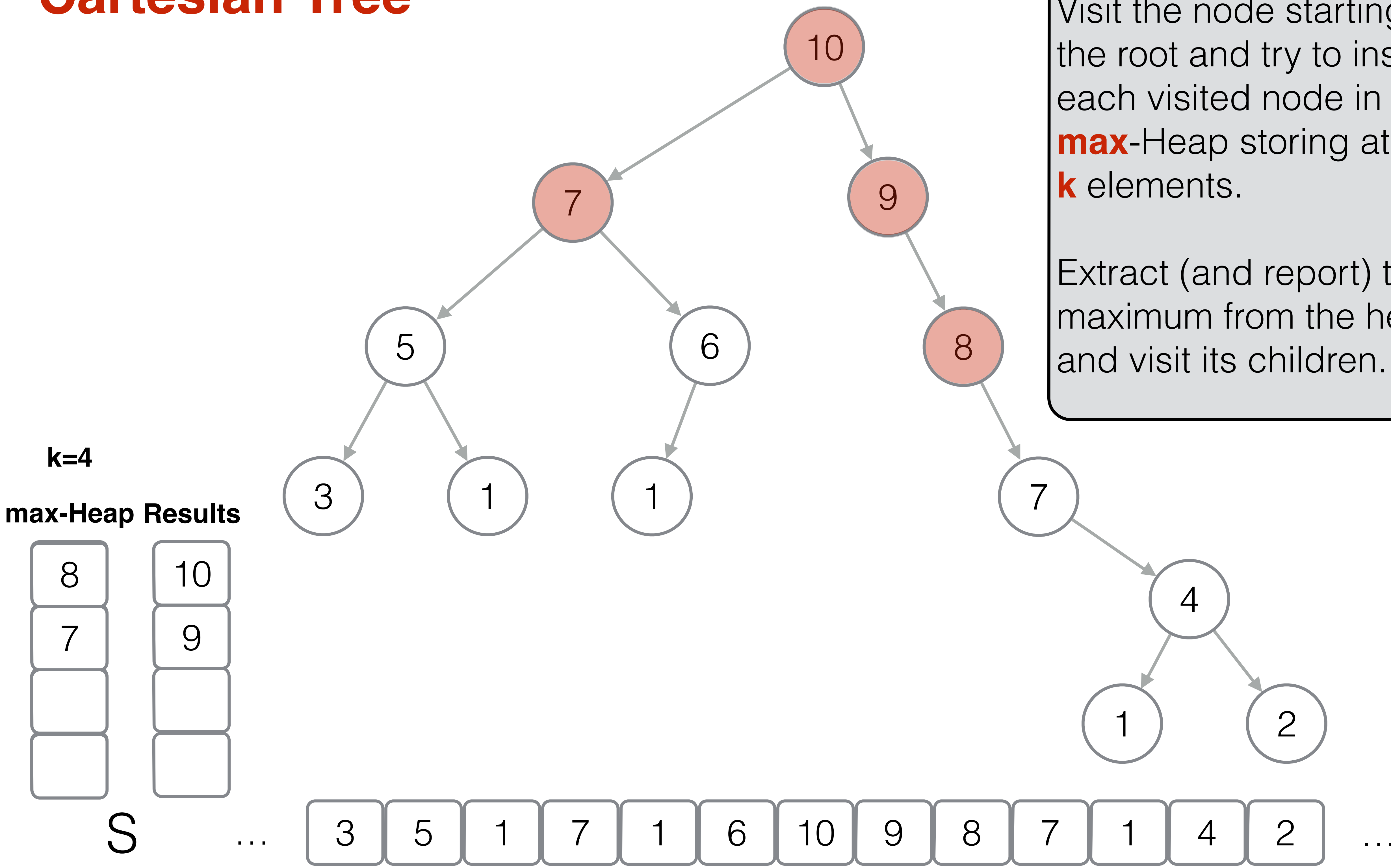
Finding Top-k

How to find Top-k?

Visit the node starting from the root and try to insert each visited node in a **max**-Heap storing at most **k** elements.

Extract (and report) the maximum from the heap and visit its children.

Cartesian Tree



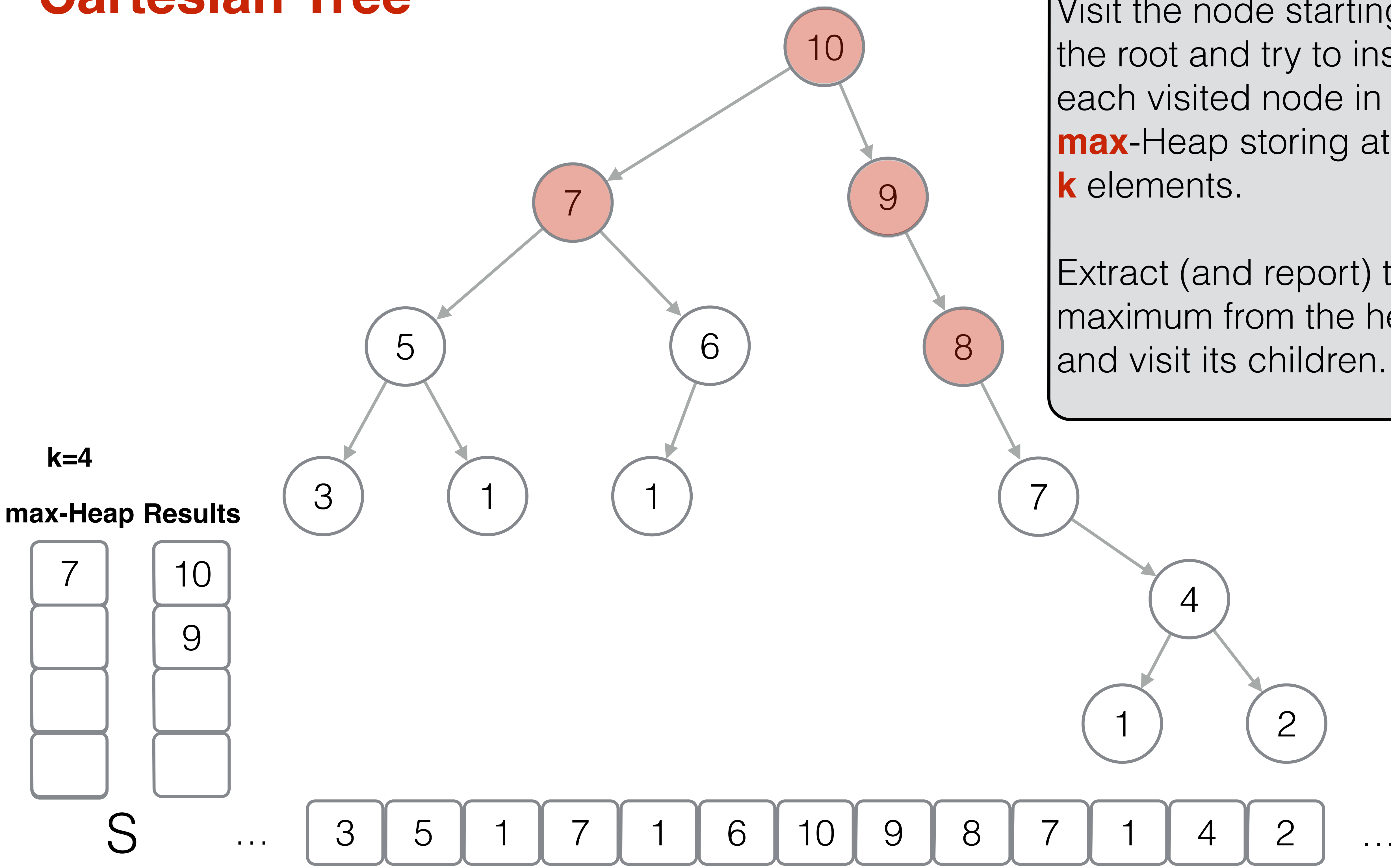
Finding Top-k

Cartesian Tree

How to find Top-k?

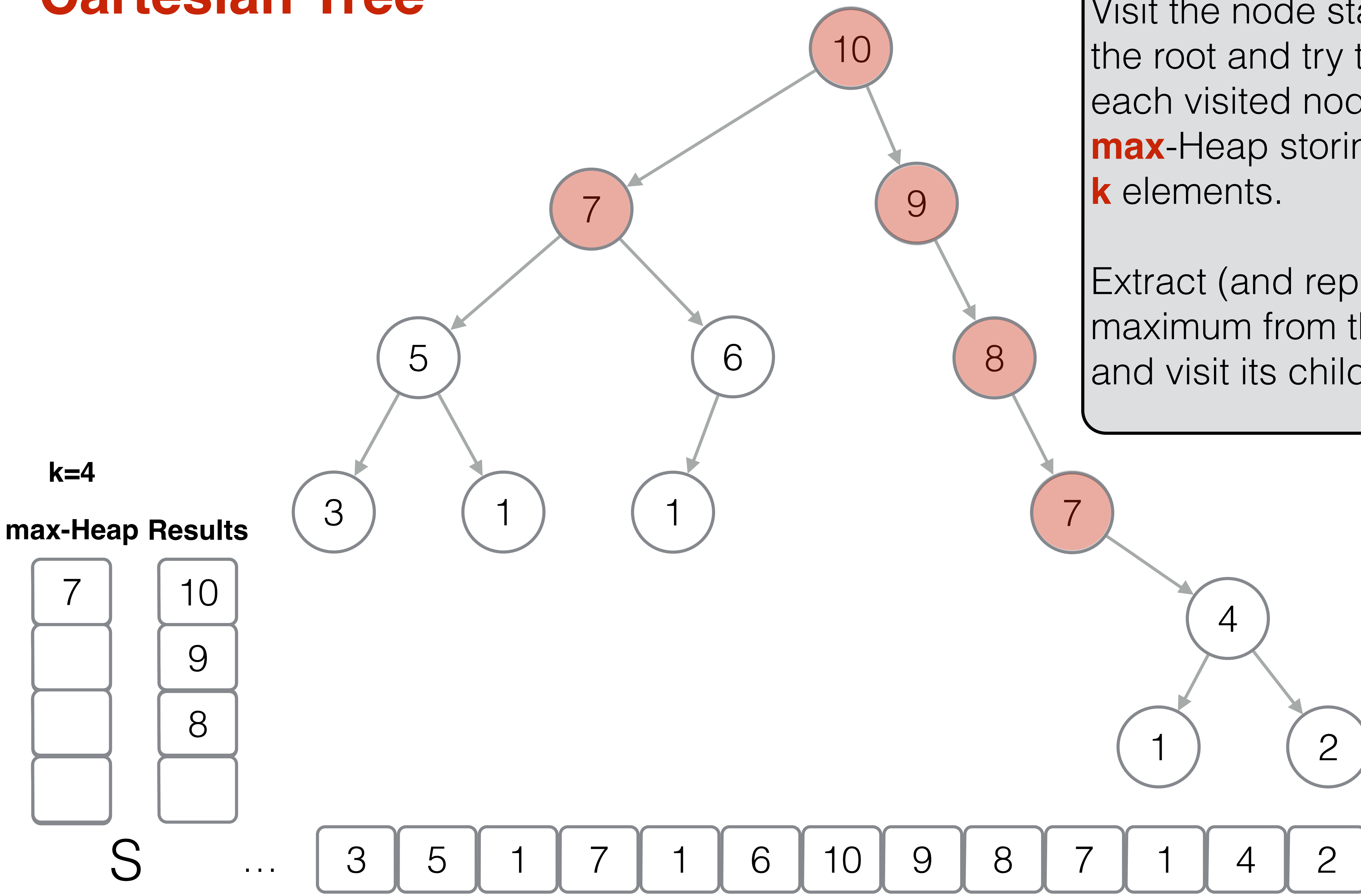
Visit the node starting from the root and try to insert each visited node in a **max**-Heap storing at most **k** elements.

Extract (and report) the maximum from the heap and visit its children.



Finding Top-k

Cartesian Tree



How to find Top-k?

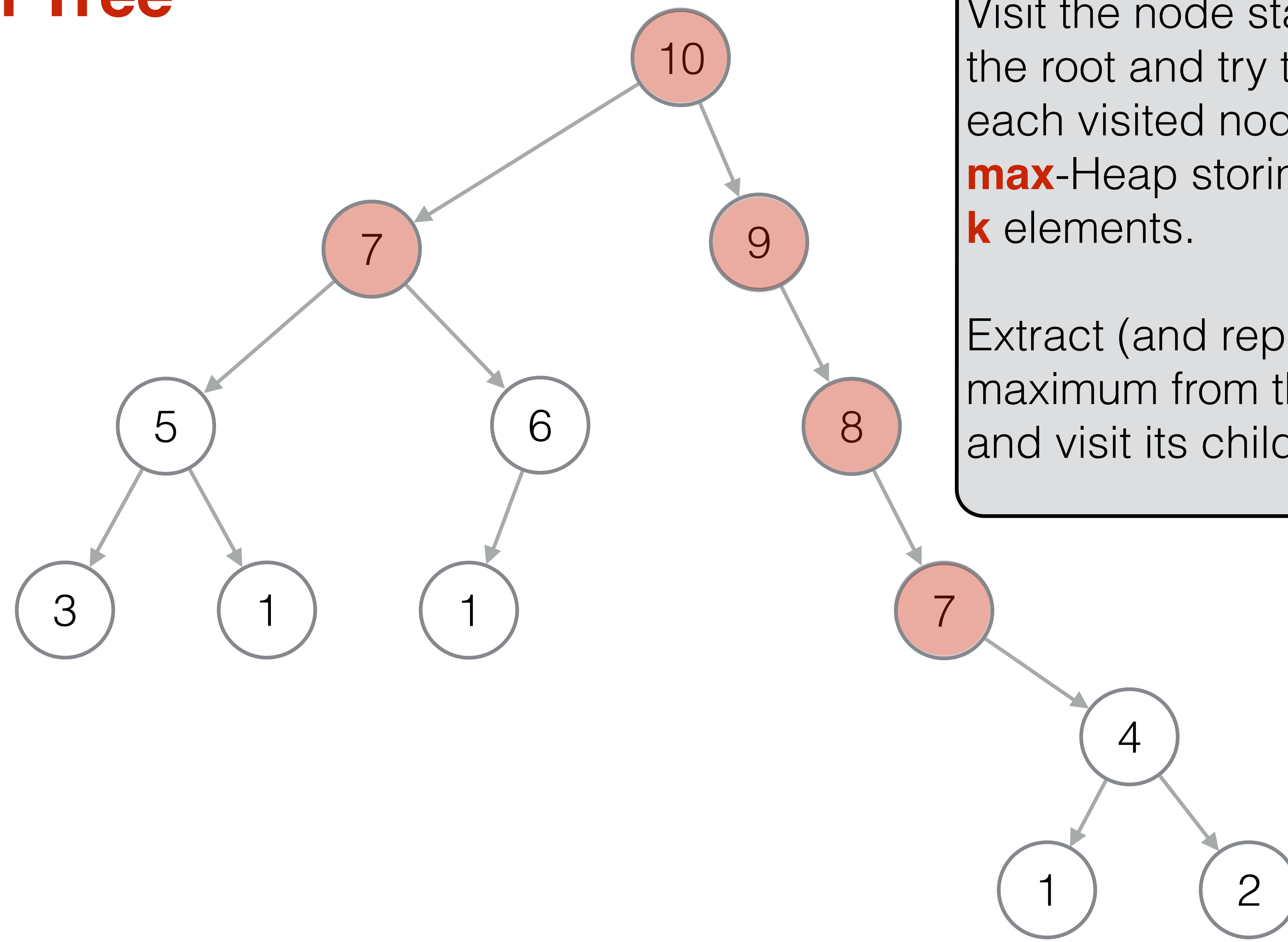
Visit the node starting from the root and try to insert each visited node in a **max**-Heap storing at most **k** elements.

Extract (and report) the maximum from the heap and visit its children.

Finding Top-k

How to find Top-k?

Cartesian Tree



Visit the node starting from the root and try to insert each visited node in a **max**-Heap storing at most **k** elements.

Extract (and report) the maximum from the heap and visit its children.

k=4

max-Heap	**Results**
7	10
7	9
	8

S

…	3	5	1	7	1	6	10	9	8	7	1	4	2	…

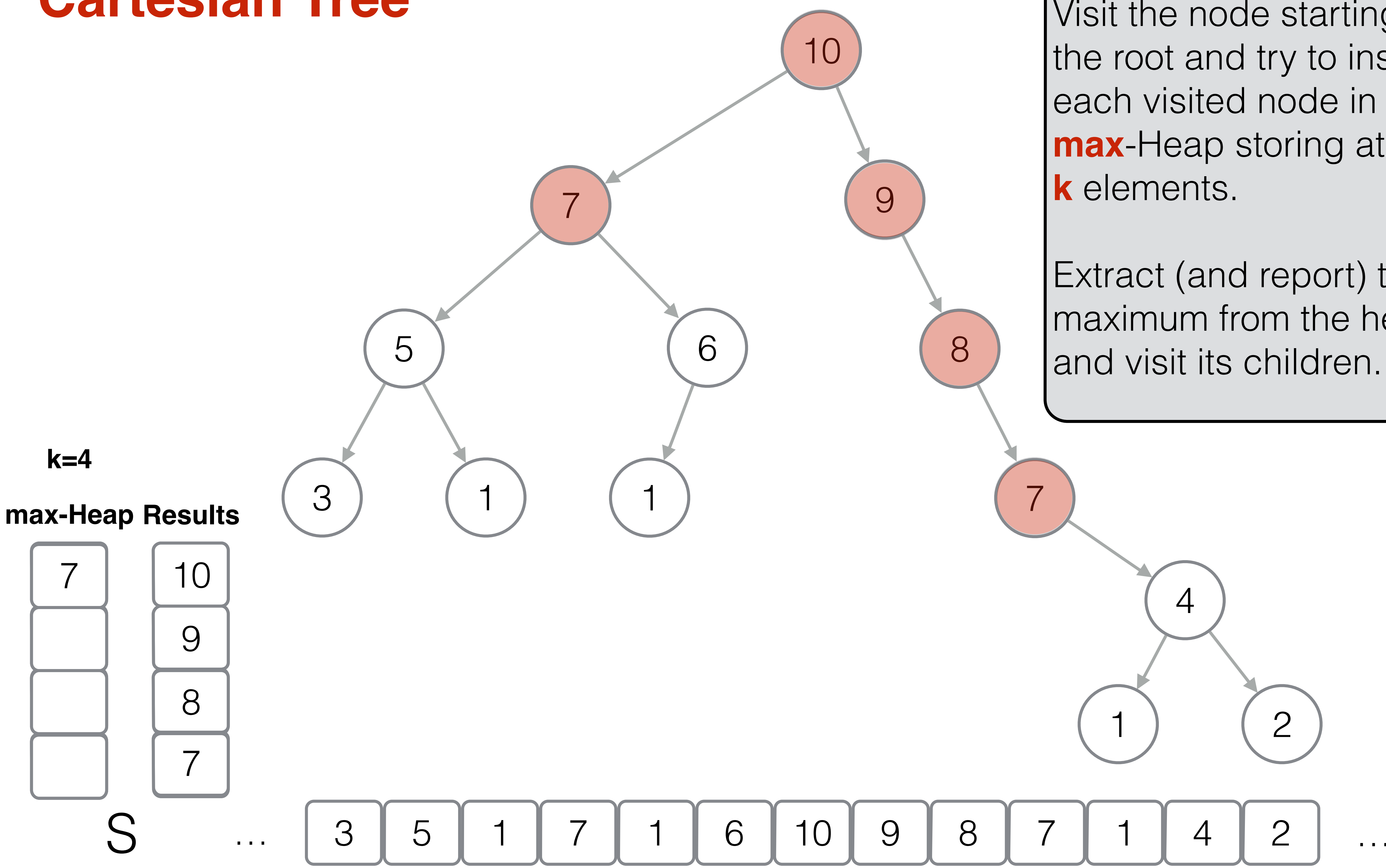
Finding Top-k

How to find Top-k?

Visit the node starting from the root and try to insert each visited node in a **max**-Heap storing at most **k** elements.

Extract (and report) the maximum from the heap and visit its children.

Cartesian Tree



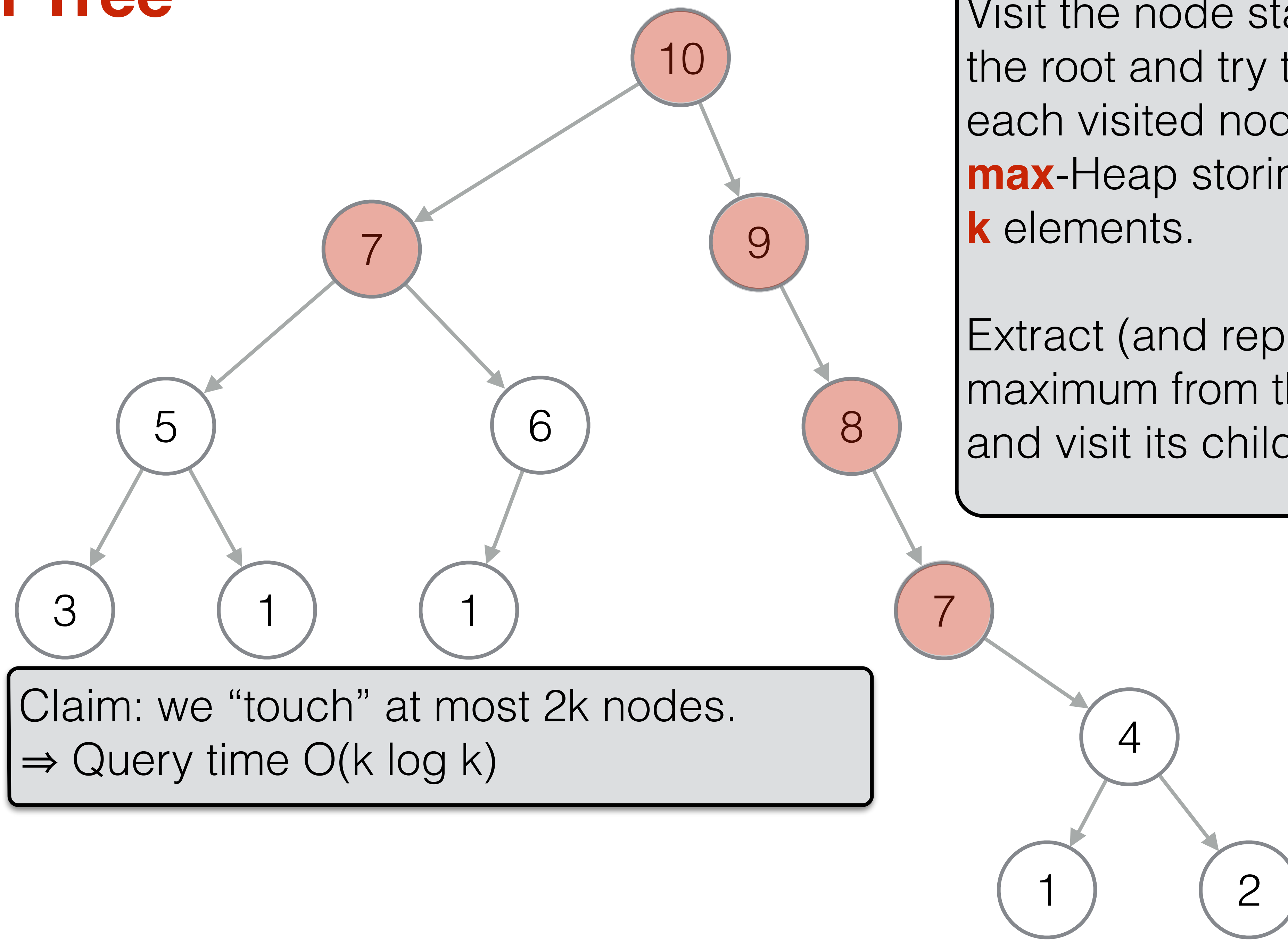
Finding Top-k

How to find Top-k?

Cartesian Tree

Visit the node starting from the root and try to insert each visited node in a **max**-Heap storing at most **k** elements.

Extract (and report) the maximum from the heap and visit its children.



Claim: we "touch" at most 2k nodes.
⇒ Query time O(k log k)

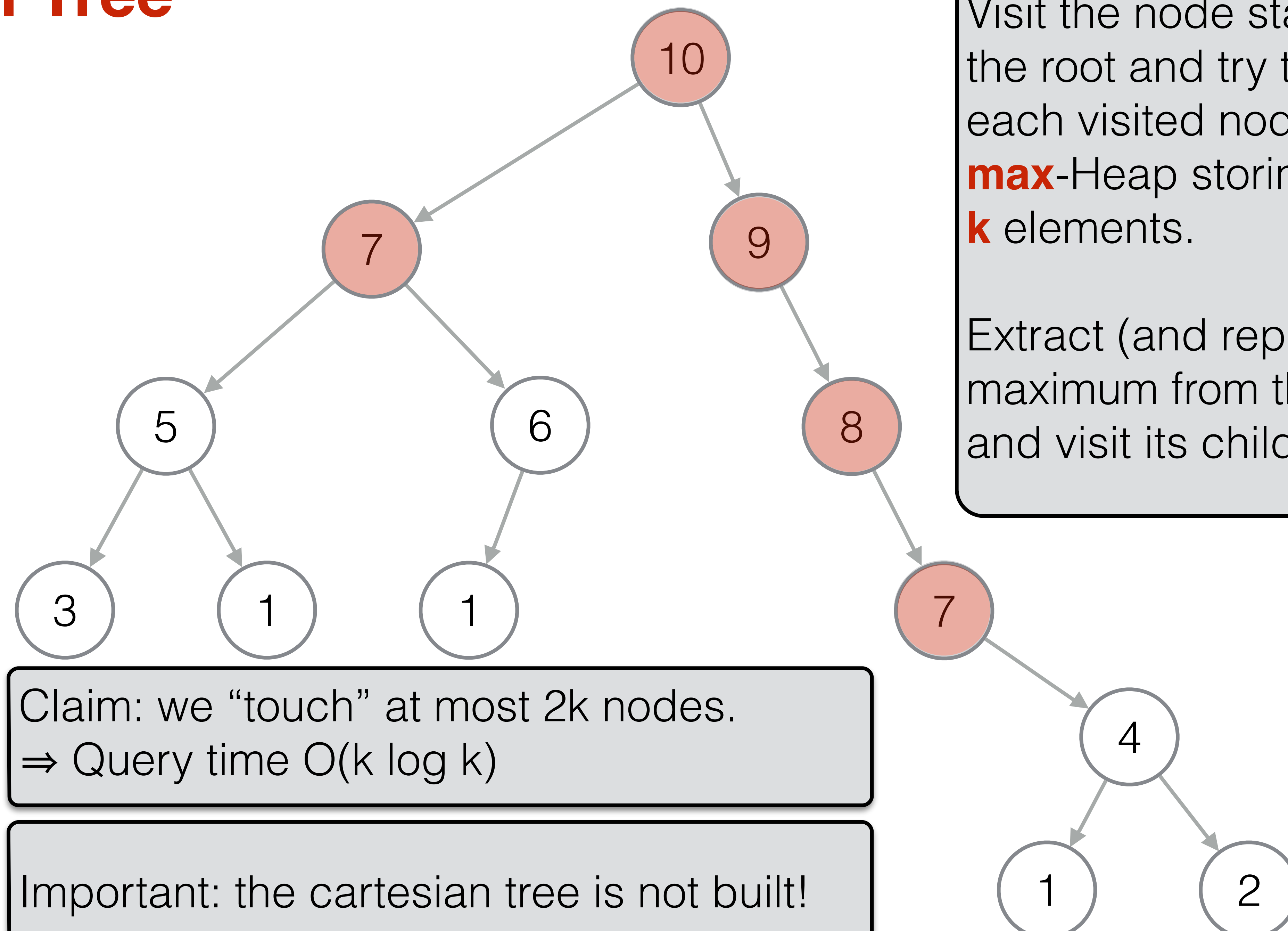
k=4

max-Heap	**Results**
7	10
	9
	8
	7

S … | 3 | 5 | 1 | 7 | 1 | 6 | 10 | 9 | 8 | 7 | 1 | 4 | 2 | …

Finding Top-k

Cartesian Tree



How to find Top-k?

Visit the node starting from the root and try to insert each visited node in a **max**-Heap storing at most **k** elements.

Extract (and report) the maximum from the heap and visit its children.

Claim: we “touch” at most 2k nodes.
⇒ Query time O(k log k)

Important: the cartesian tree is not built!

k=4

max-Heap	**Results**
7	10
	9
	8
	7

S

…	3	5	1	7	1	6	10	9	8	7	1	4	2	…

Finding Top-k

Cartesian Tree

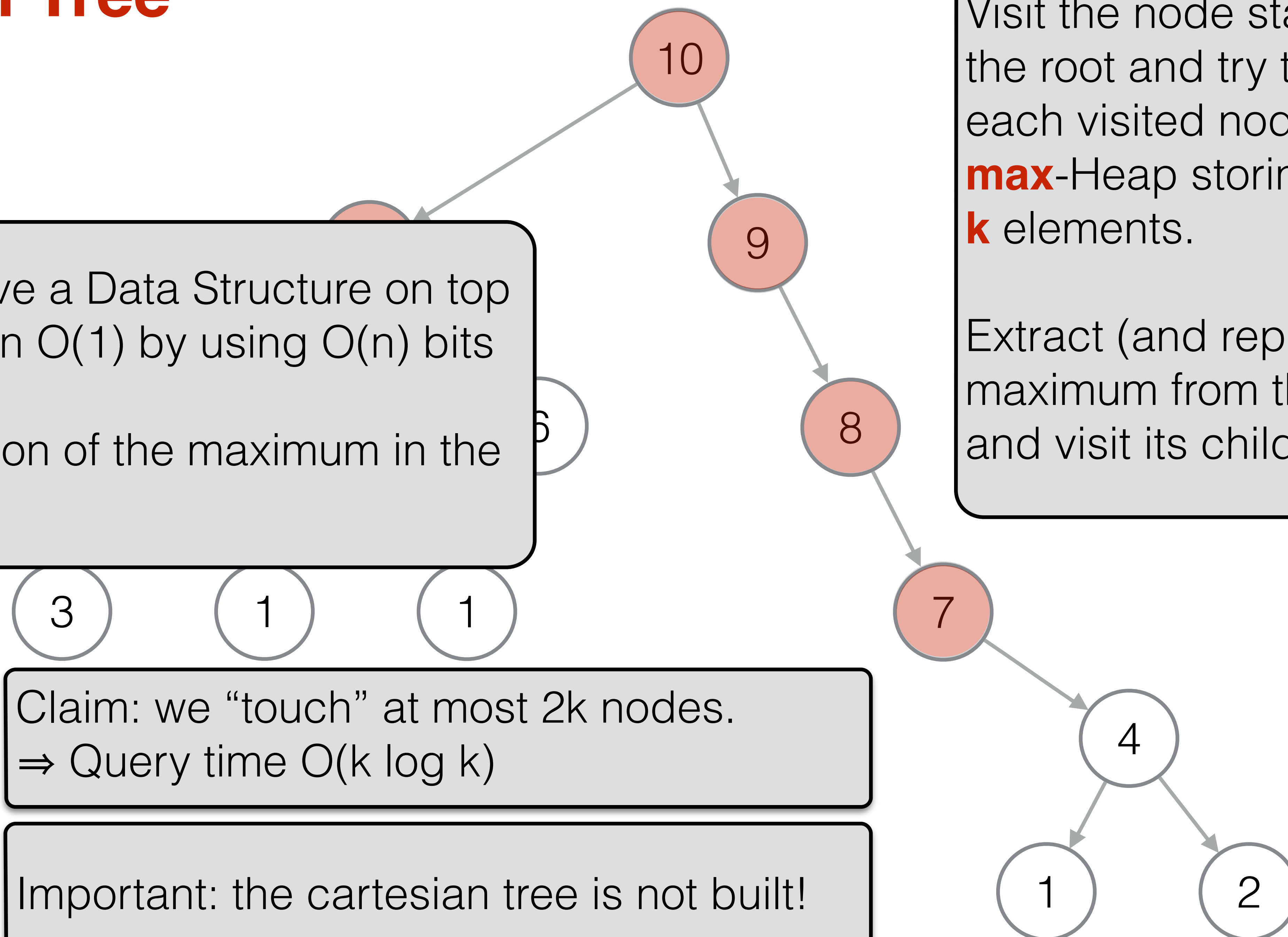
How to find Top-k?

Visit the node starting from the root and try to insert each visited node in a **max**-Heap storing at most **k** elements.

Extract (and report) the maximum from the heap and visit its children.

Assume you have a Data Structure on top of S answering in O(1) by using O(n) bits

RMQ(i,j) = position of the maximum in the range S[i,j]



Claim: we "touch" at most 2k nodes.
⇒ Query time O(k log k)

Important: the cartesian tree is not built!

max-Heap Results

max-Heap	Results
7	10
	9
	8
	7

S … 3 5 1 7 1 6 10 9 8 7 1 4 2 …

Range Maximum Query (1)

	0	1	2	3	4	5	6	7	8	9	10	11
S	3	5	1	7	1	6	10	9	8	7	1	4

Range Maximum Query (1)

Space: $O(n^2 \log n)$ bits
Query time: $O(1)$

	0	1	2	3	4	5	6	7	8	9	10	11
S	3	5	1	7	1	6	10	9	8	7	1	4

Range Maximum Query (1)

Space: $O(n^2 \log n)$ bits
Query time: $O(1)$

Precompute the answer to any possible query.

There are $O(n^2)$ distinct queries!

	0	1	2	3	4	5	6	7	8	9	10	11
S	3	5	1	7	1	6	10	9	8	7	1	4

Range Maximum Query (1)

Space: $O(n^2 \log n)$ bits
Query time: $O(1)$

$M[i,j] = RMQ(i,j)$

Precompute the answer to any possible query.

There are $O(n^2)$ distinct queries!

	0	1	2	3	4	5	6	7	8	9	10	11
S	3	5	1	7	1	6	10	9	8	7	1	4

Range Maximum Query (1)

Space: $O(n^2 \log n)$ bits
Query time: $O(1)$

Precompute the answer to any possible query.

There are $O(n^2)$ distinct queries!

$M[i,j] = RMQ(i,j)$

M	0	1	2	3	4	5	6	7	8	9	10	11
0												
1												
2												
3												
4												
5												
6												
7												
8												
9												
10												
11												

	0	1	2	3	4	5	6	7	8	9	10	11
S	3	5	1	7	1	6	10	9	8	7	1	4

Range Maximum Query (1)

Space: $O(n^2 \log n)$ bits
Query time: $O(1)$

$$M[i,j] = RMQ(i,j)$$

Precompute the answer to any possible query.

There are $O(n^2)$ distinct queries!

M

	0	1	2	3	4	5	6	7	8	9	10	11
0												
1												
2						3						
3												
4												
5												
6												
7												
8												
9												
10												
11												

	0	1	2	3	4	5	6	7	8	9	10	11
S	3	5	1	7	1	6	10	9	8	7	1	4

Range Maximum Query (2)

	0	1	2	3	4	5	6	7	8	9	10	11
S	3	5	1	7	1	6	10	9	8	7	1	4

Range Maximum Query (2)

Space: $O(n \log^2 n)$ bits
Query time: $O(1)$

	0	1	2	3	4	5	6	7	8	9	10	11
S	3	5	1	7	1	6	10	9	8	7	1	4

Range Maximum Query (2)

Space: $O(n \log^2 n)$ bits
Query time: $O(1)$

Maximum in a interval is the max between the maxima of any its subintervals

	0	1	2	3	4	5	6	7	8	9	10	11
S	3	5	1	7	1	6	10	9	8	7	1	4

Range Maximum Query (2)

Space: $O(n \log^2 n)$ bits
Query time: $O(1)$

Maximum in a interval is the max between the maxima of any its subintervals

Precompute the answer to every interval of size a power of 2.

There are $O(\log n)$ possible intervals starting at any position i.

	0	1	2	3	4	5	6	7	8	9	10	11
S	3	5	1	7	1	6	10	9	8	7	1	4

Range Maximum Query (2)

Space: $O(n \log^2 n)$ bits
Query time: $O(1)$

Maximum in a interval is the max between the maxima of any its subintervals

Precompute the answer to every interval of size a power of 2.

There are $O(\log n)$ possible intervals starting at any position i.

$M[i,j] = RMQ(i,i+2^j)$

M

	0	1	2	3	4
0					
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					

	0	1	2	3	4	5	6	7	8	9	10	11
S	3	5	1	7	1	6	10	9	8	7	1	4

Range Maximum Query (2)

Space: $O(n \log^2 n)$ bits
Query time: $O(1)$

Maximum in a interval is the max between the maxima of any its subintervals

Precompute the answer to every interval of size a power of 2.

There are $O(\log n)$ possible intervals starting at any position i.

$$M[i,j] = RMQ(i, i+2^j)$$

M

	0	1	2	3	4
0					
1				?	
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					

	0	1	2	3	4	5	6	7	8	9	10	11
S	3	5	1	7	1	6	10	9	8	7	1	4

Range Maximum Query (2)

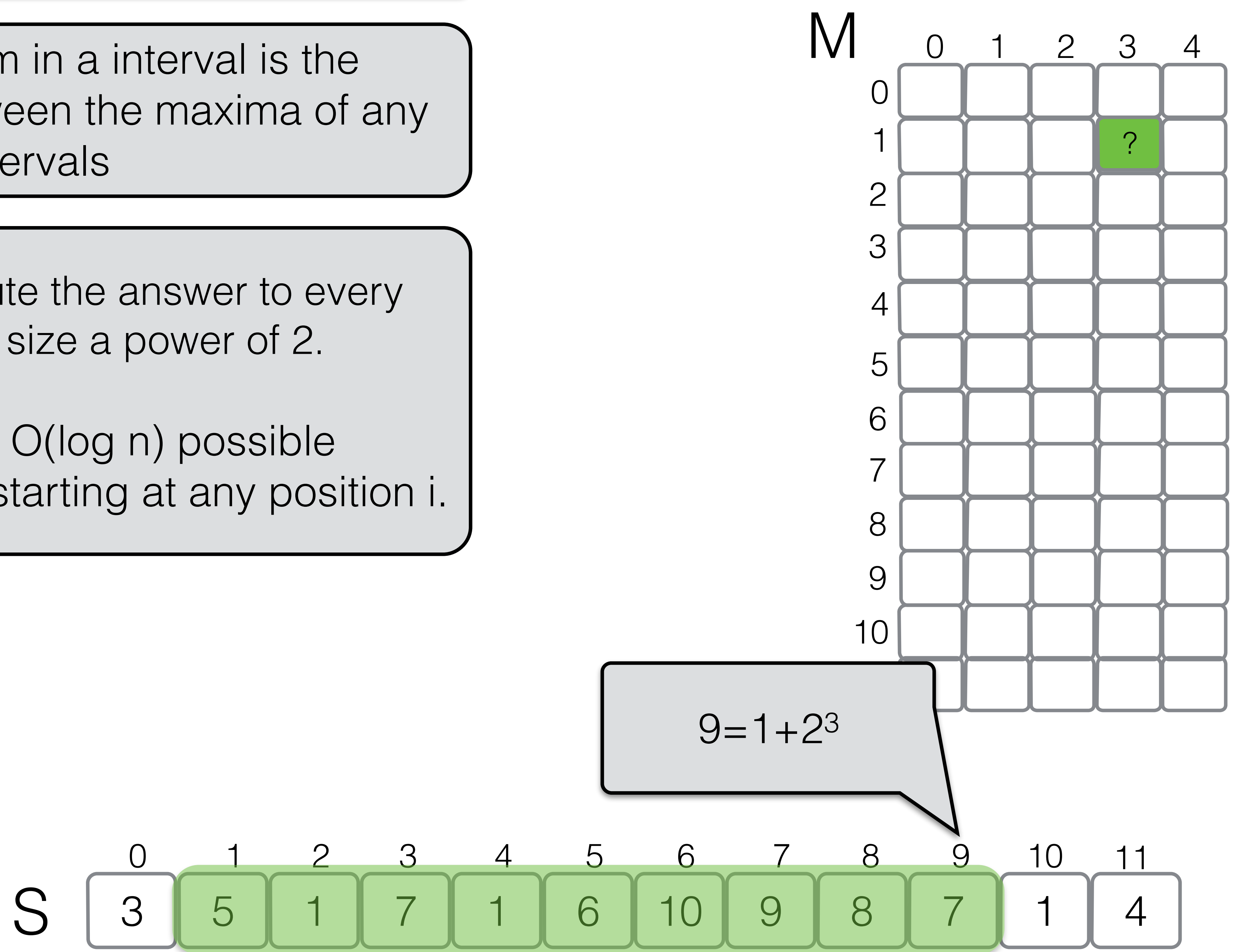
Space: $O(n \log^2 n)$ bits
Query time: $O(1)$

Maximum in a interval is the max between the maxima of any its subintervals

Precompute the answer to every interval of size a power of 2.

There are $O(\log n)$ possible intervals starting at any position i.

$$M[i,j] = RMQ(i,i+2^j)$$



Range Maximum Query (2)

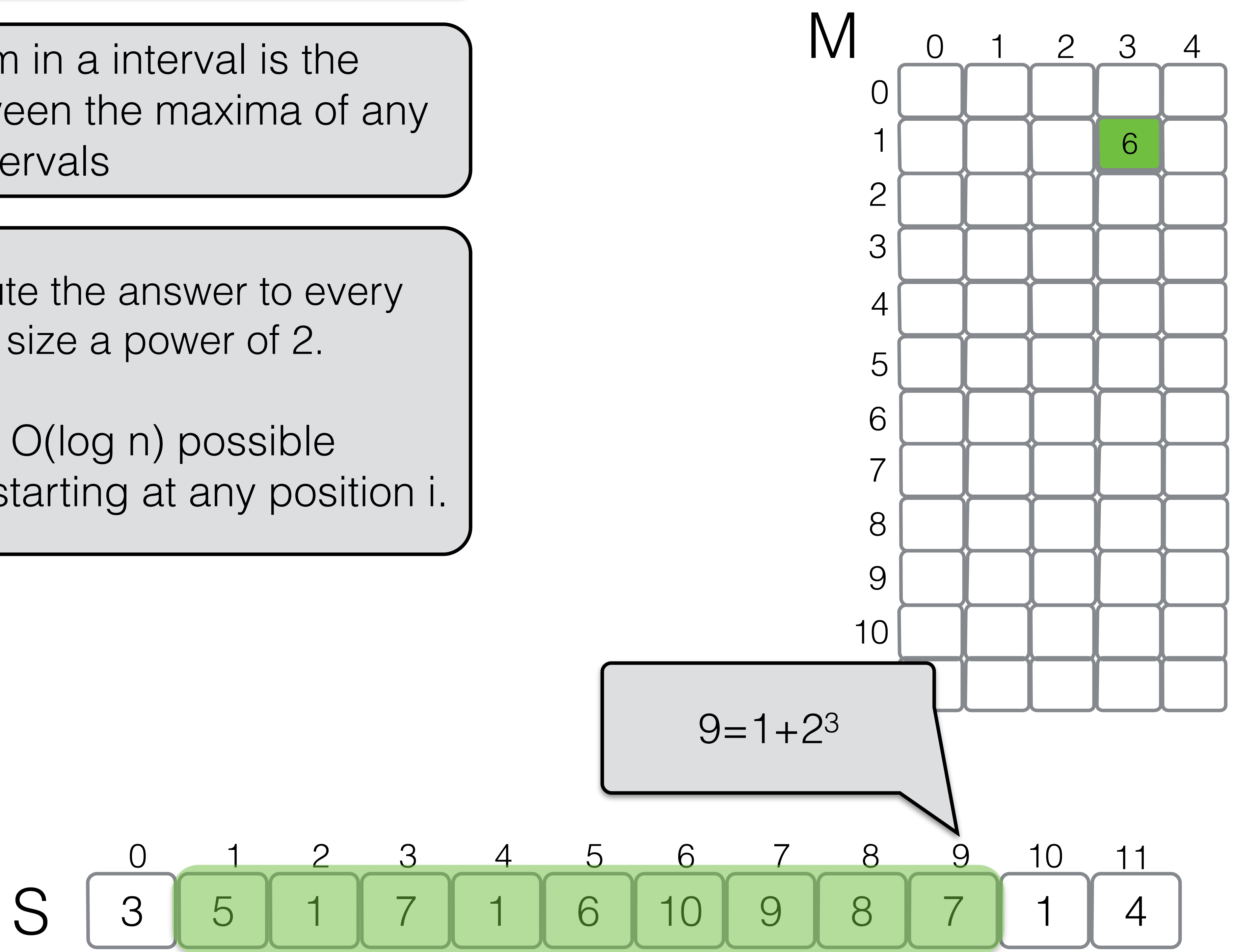
Space: $O(n \log^2 n)$ bits
Query time: $O(1)$

Maximum in a interval is the max between the maxima of any its subintervals

Precompute the answer to every interval of size a power of 2.

There are $O(\log n)$ possible intervals starting at any position i.

$$M[i,j] = RMQ(i,i+2^j)$$



Range Maximum Query (2)

Space: $O(n \log^2 n)$ bits
Query time: $O(1)$

Maximum of a interval is the max between the maxima of any its subintervals

Precompute the answer to every interval of size a power of 2.

There are $O(\log n)$ possible intervals starting at any position i.

$$M[i,j] = RMQ(i,i+2^j)$$

M

	0	1	2	3	4
0					
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					

S

0	1	2	3	4	5	6	7	8	9	10	11
3	5	1	7	1	6	10	9	8	7	1	4

Range Maximum Query (2)

Space: $O(n \log^2 n)$ bits
Query time: $O(1)$

Maximum of a interval is the max between the maxima of any its subintervals

Precompute the answer to every interval of size a power of 2.

There are $O(\log n)$ possible intervals starting at any position i.

RMQ(1,7) =

$$M[i,j] = RMQ(i,i+2^j)$$

M

	0	1	2	3	4
0					
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					

S

0	1	2	3	4	5	6	7	8	9	10	11
3	5	1	7	1	6	10	9	8	7	1	4

Range Maximum Query (2)

Space: $O(n \log^2 n)$ bits
Query time: $O(1)$

Maximum of a interval is the max between the maxima of any its subintervals

Precompute the answer to every interval of size a power of 2.

There are $O(\log n)$ possible intervals starting at any position i.

$$M[i,j] = RMQ(i,i+2^j)$$

M

	0	1	2	3	4
0					
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					

$RMQ(1,7) = \text{argmax}(S[M[1,1+2^2]], S[M[7-2^2,7]]) = 6$

	0	1	2	3	4	5	6	7	8	9	10	11
S	3	5	1	7	1	6	10	9	8	7	1	4

Range Maximum Query (2)

Space: $O(n \log^2 n)$ bits
Query time: $O(1)$

Maximum of a interval is the max between the maxima of any its subintervals

Precompute the answer to every interval of size a power of 2.

There are $O(\log n)$ possible intervals starting at any position i.

$$M[i,j] = RMQ(i,i+2^j)$$

M

	0	1	2	3	4
0					
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					

$RMQ(1,7) = \text{argmax}(S[M[1,1+2^2]], S[M[7-2^2,7]]) = 6$

	0	1	2	3	4	5	6	7	8	9	10	11
S	3	5	1	7	1	6	10	9	8	7	1	4

Range Maximum Query (2)

Space: $O(n \log^2 n)$ bits
Query time: $O(1)$

Maximum of a interval is the max between the maxima of any its subintervals

Precompute the answer to every interval of size a power of 2.

There are $O(\log n)$ possible intervals starting at any position i.

$$M[i,j] = RMQ(i,i+2^j)$$

M

	0	1	2	3	4
0					
1			3		
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					

$RMQ(1,7) = \text{argmax}(S[M[1,1+2^2]], S[M[7-2^2,7]]) = 6$

	0	1	2	3	4	5	6	7	8	9	10	11
S	3	5	1	7	1	6	10	9	8	7	1	4

Range Maximum Query (2)

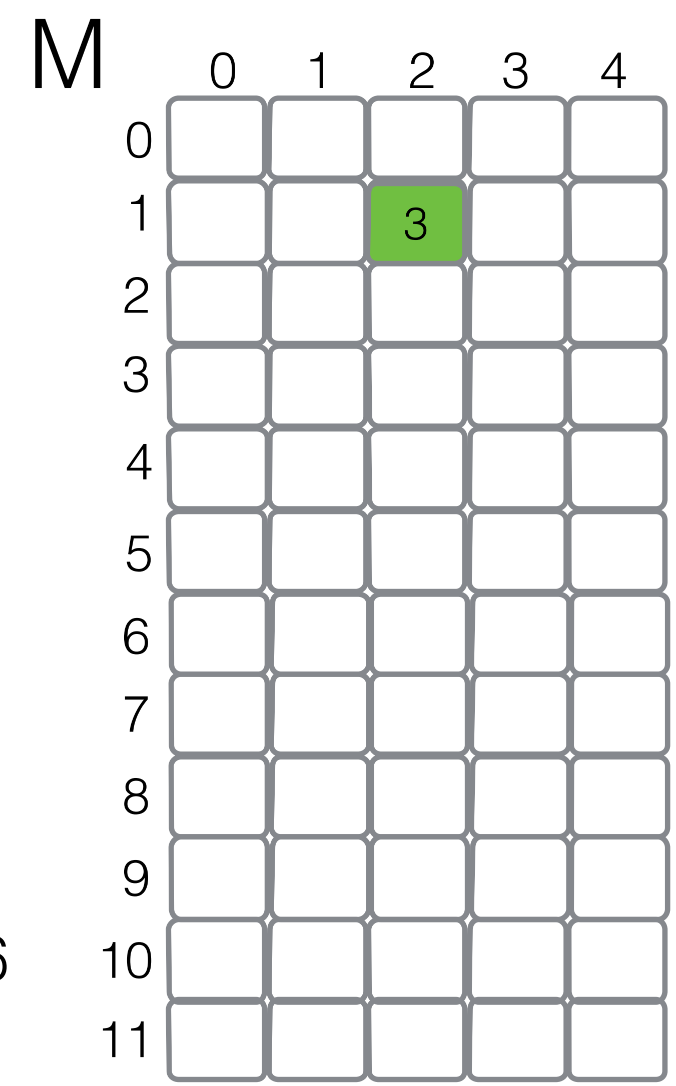
Space: $O(n \log^2 n)$ bits
Query time: $O(1)$

Maximum of a interval is the max between the maxima of any its subintervals

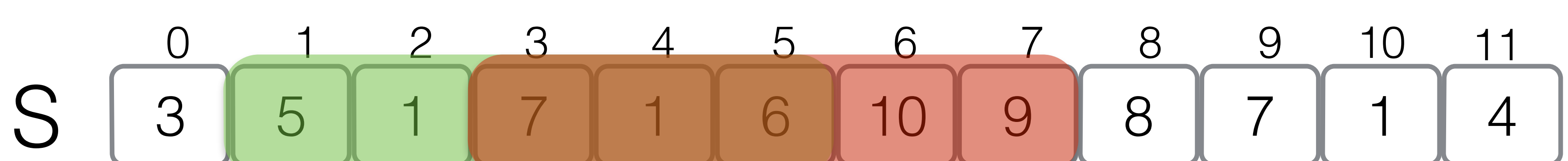
Precompute the answer to every interval of size a power of 2.

There are $O(\log n)$ possible intervals starting at any position i.

$$M[i,j] = RMQ(i,i+2^j)$$



$$RMQ(1,7) = \text{argmax}(S[M[1,1+2^2]], S[M[7-2^2,7]]) = 6$$



Range Maximum Query (2)

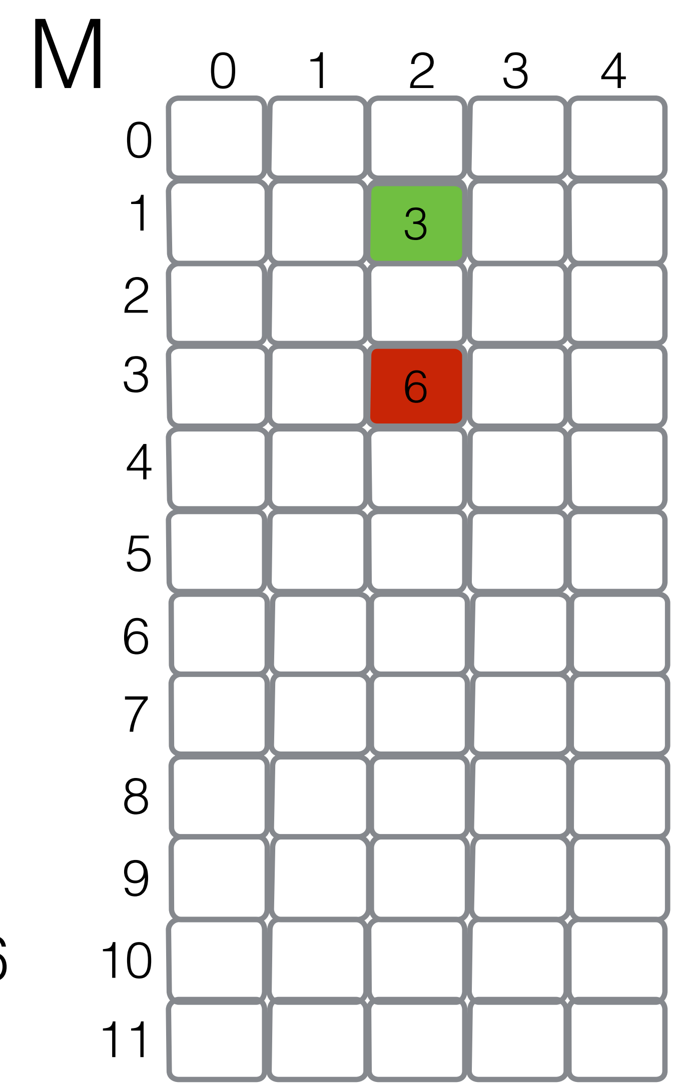
Space: $O(n \log^2 n)$ bits
Query time: $O(1)$

Maximum of a interval is the max between the maxima of any its subintervals

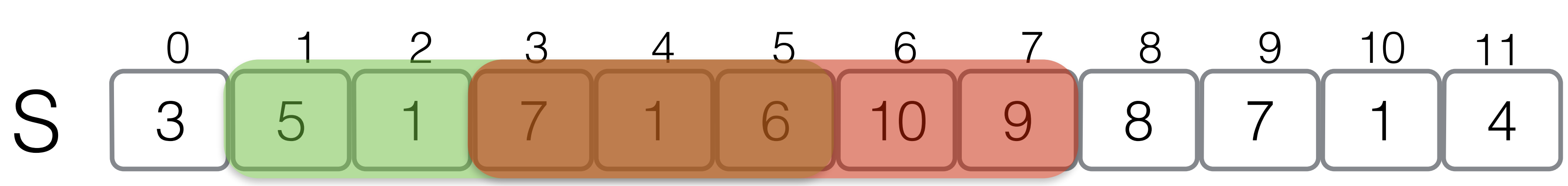
Precompute the answer to every interval of size a power of 2.

There are $O(\log n)$ possible intervals starting at any position i.

$$M[i,j] = RMQ(i,i+2^j)$$



$RMQ(1,7) = argmax(S[M[1,1+2^2]], S[M[7-2^2,7]]) = 6$



Range Maximum Query (2)

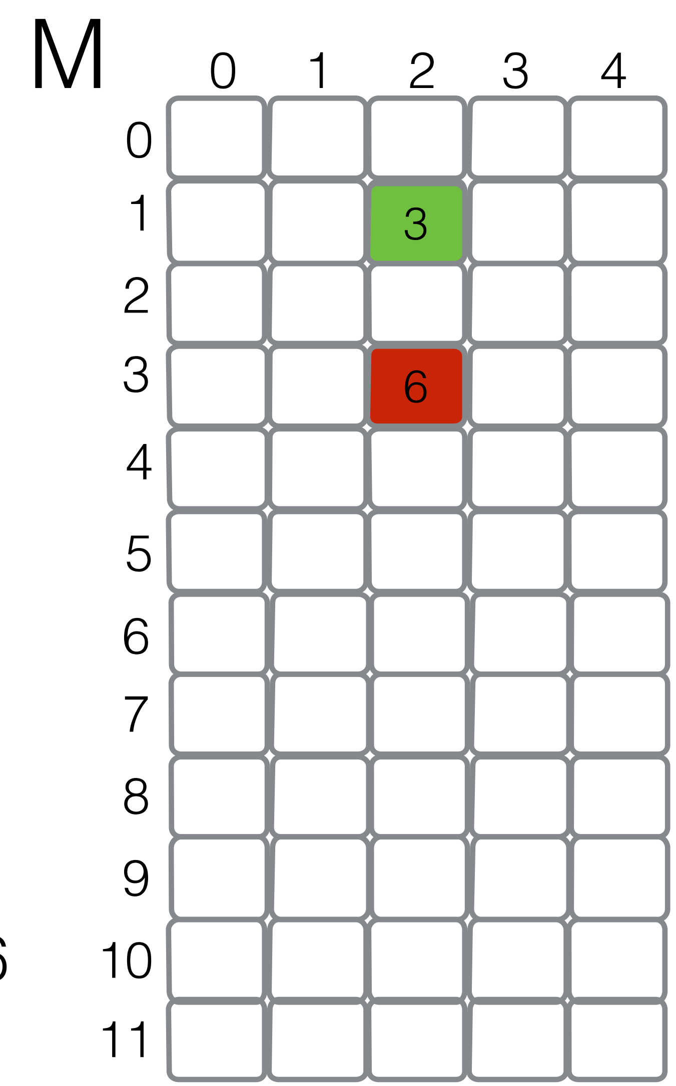
Space: $O(n \log^2 n)$ bits
Query time: $O(1)$

Maximum of a interval is the max between the maxima of any its subintervals

Precompute the answer to every interval of size a power of 2.

There are $O(\log n)$ possible intervals starting at any position i.

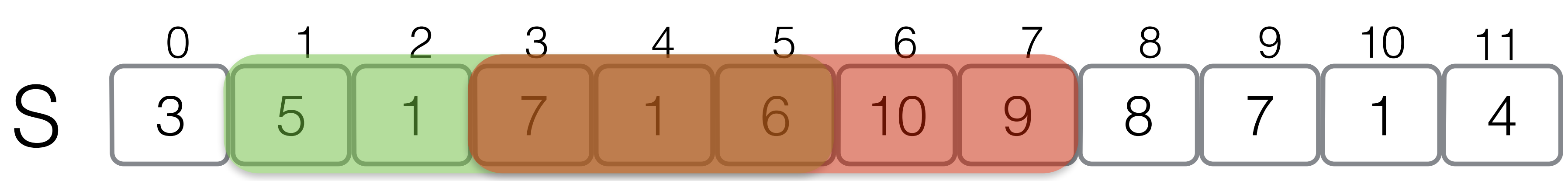
$$M[i,j] = RMQ(i,i+2^j)$$



$$RMQ(1,7) = \text{argmax}(S[M[1,1+2^2]], S[M[7-2^2,7]]) = 6$$

$$RMQ(i,j) = \text{argmax}(S[M[i,i+2^{len}]], S[M[j-2^{len},j]])$$

where $len = \lfloor \log (j-i+1) \rfloor$



Range Maximum Query (3)

	0	1	2	3	4	5	6	7	8	9	10	11
S	3	5	1	7	1	6	10	9	8	7	1	4

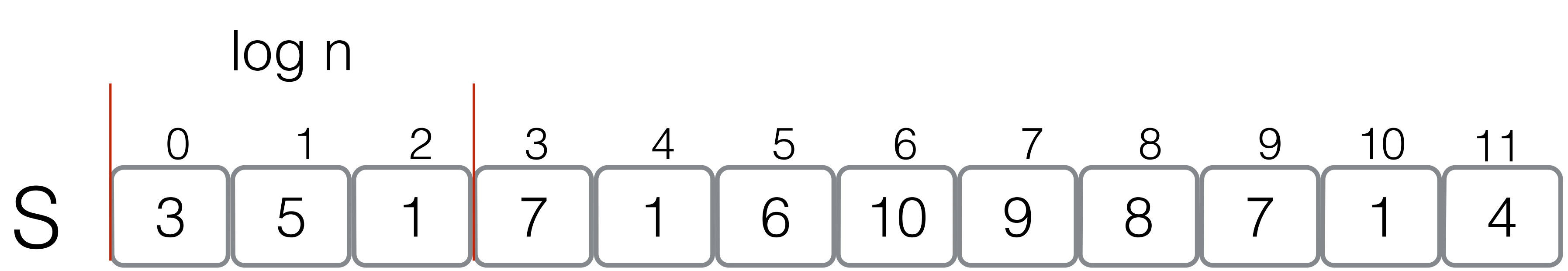
Range Maximum Query (3)

Space: O(n log n) bits
Query time: O(log n)

	0	1	2	3	4	5	6	7	8	9	10	11
S	3	5	1	7	1	6	10	9	8	7	1	4

Range Maximum Query (3)

Space: O(n log n) bits
Query time: O(log n)



Range Maximum Query (3)

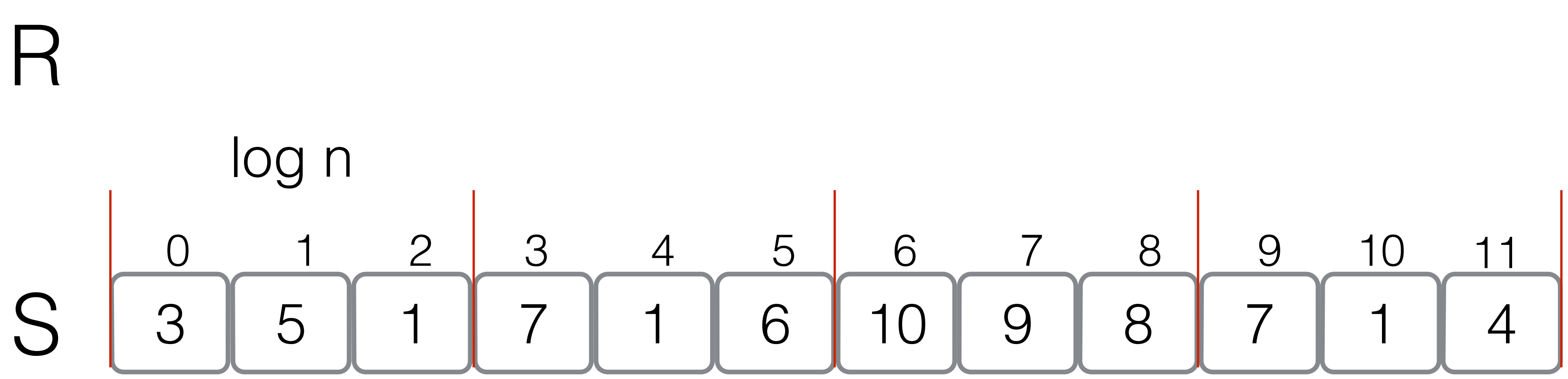
Space: O(n log n) bits
Query time: O(log n)

log n

	0	1	2	3	4	5	6	7	8	9	10	11
S	3	5	1	7	1	6	10	9	8	7	1	4

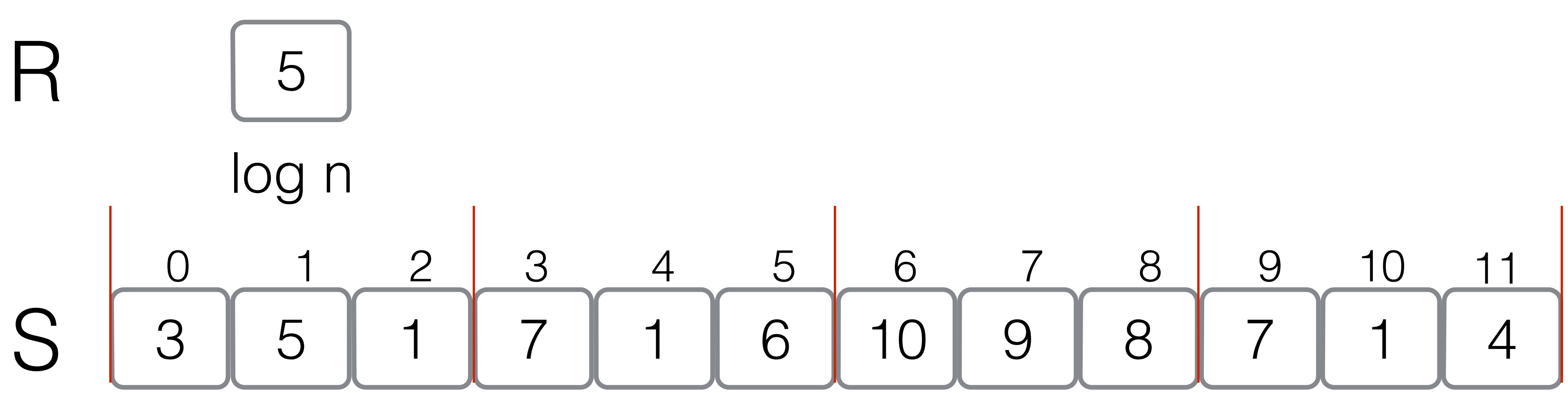
Range Maximum Query (3)

Space: O(n log n) bits
Query time: O(log n)



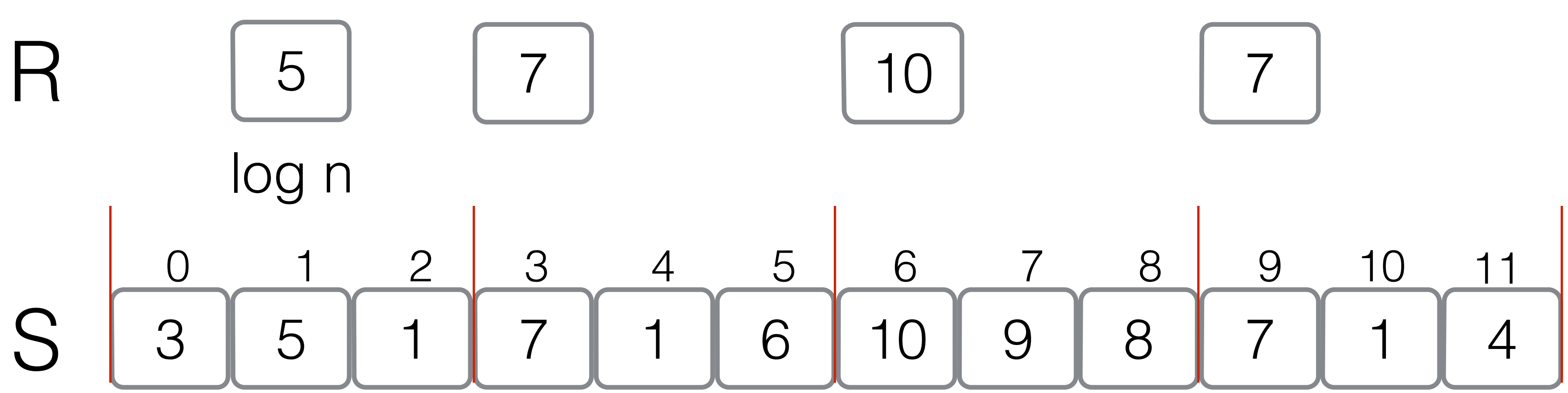
Range Maximum Query (3)

Space: O(n log n) bits
Query time: O(log n)



Range Maximum Query (3)

Space: O(n log n) bits
Query time: O(log n)

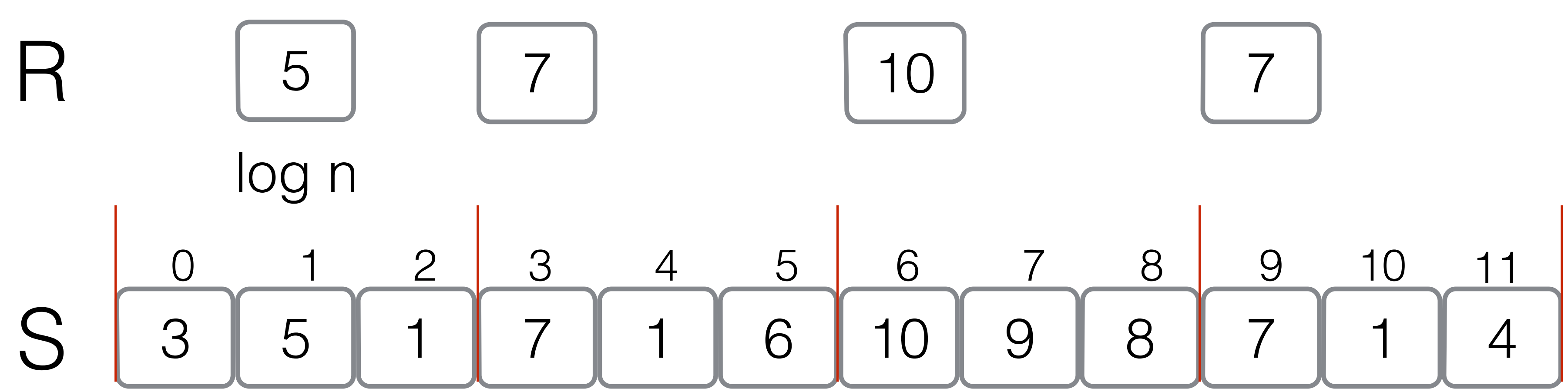


Range Maximum Query (3)

Space: O(n log n) bits
Query time: O(log n)

Use the previous solution on R!

Space: ? bits
Query time: O(1)

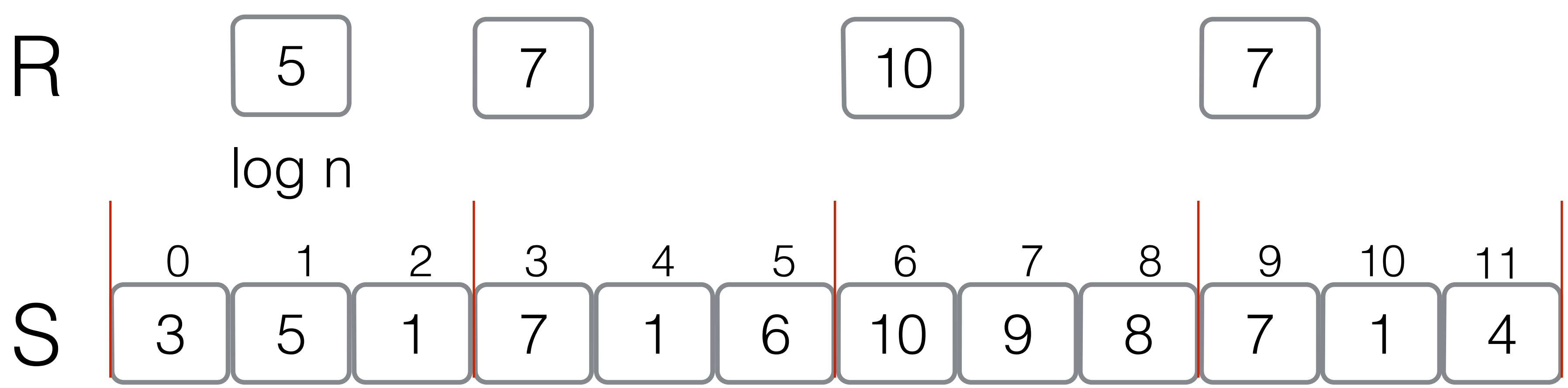


Range Maximum Query (3)

Space: O(n log n) bits
Query time: O(log n)

Use the previous solution on R!

Space: O(n log n) bits
Query time: O(1)



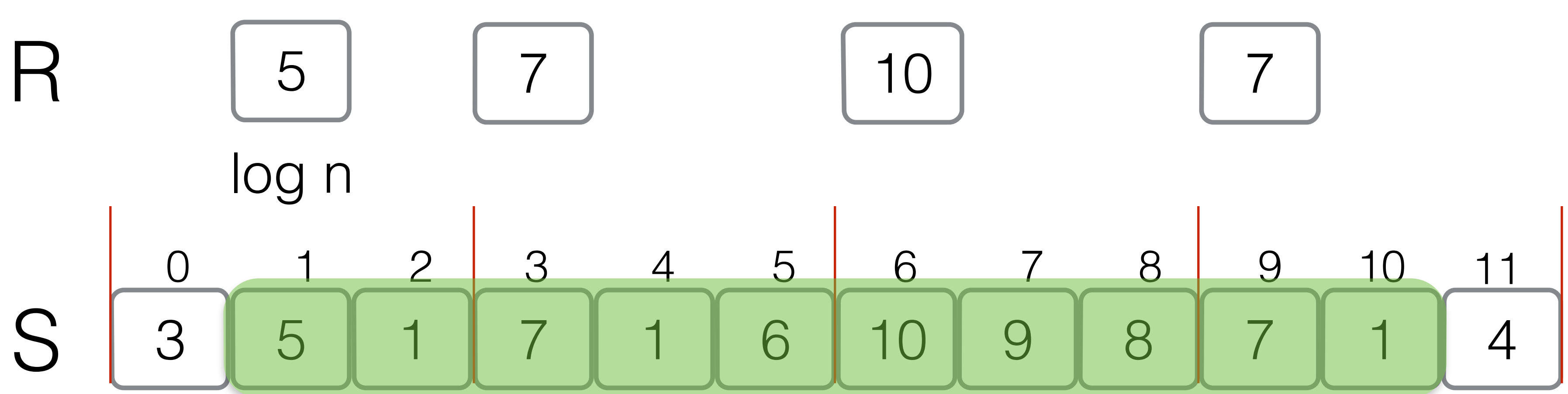
Range Maximum Query (3)

Space: O(n log n) bits
Query time: O(log n)

Use the previous solution on R!

Space: O(n log n) bits
Query time: O(1)

RMQ(1,10) = ?



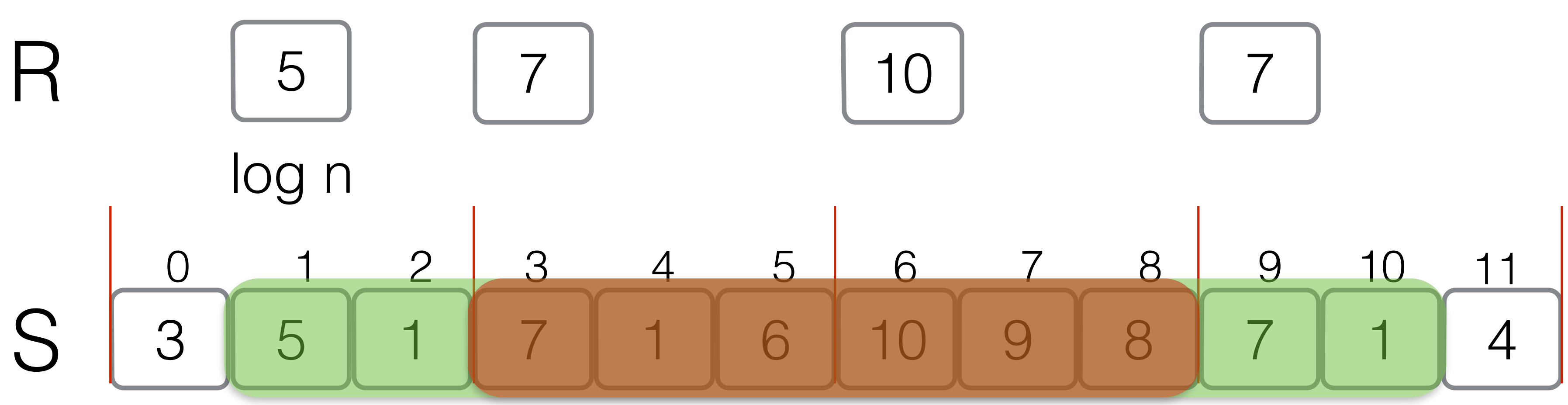
Range Maximum Query (3)

Space: O(n log n) bits
Query time: O(log n)

Use the previous solution on R!

Space: O(n log n) bits
Query time: O(1)

RMQ(1,10) = ?



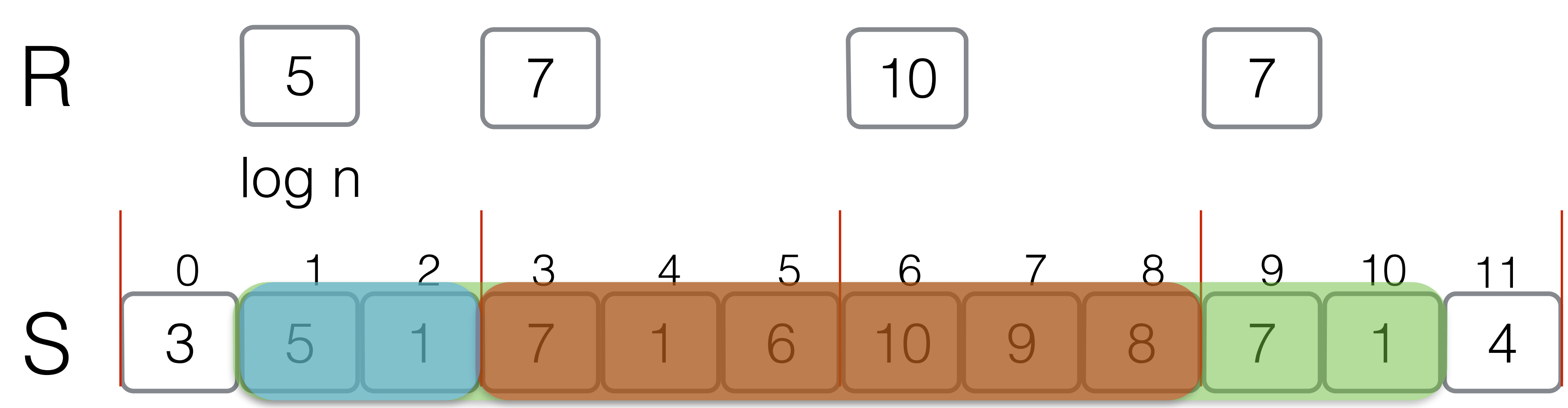
Range Maximum Query (3)

Space: O(n log n) bits
Query time: O(log n)

Use the previous solution on R!

Space: O(n log n) bits
Query time: O(1)

RMQ(1,10) = ?



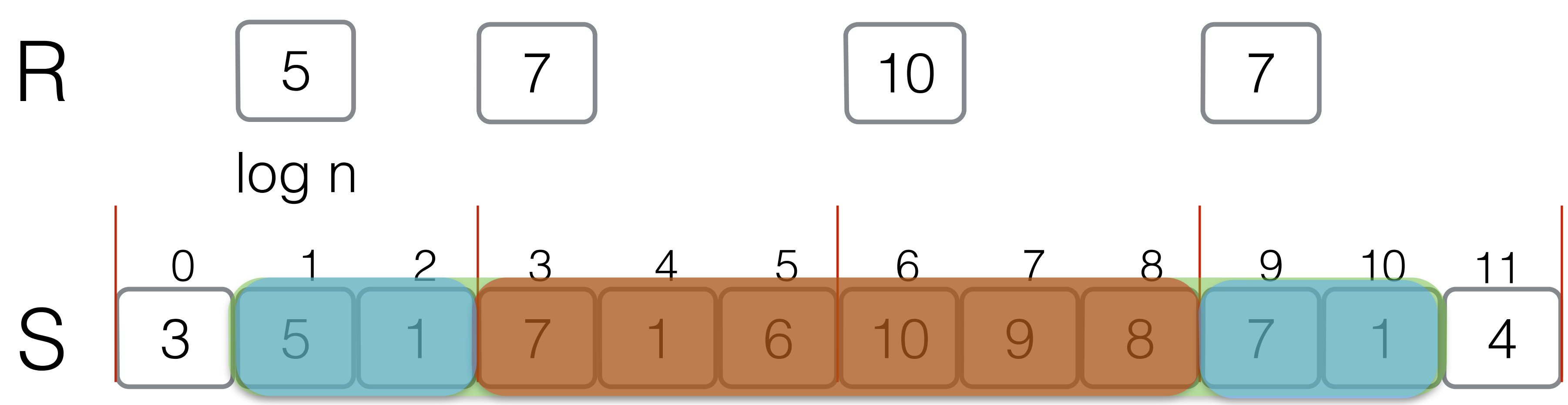
Range Maximum Query (3)

Space: O(n log n) bits
Query time: O(log n)

Use the previous solution on R!

Space: O(n log n) bits
Query time: O(1)

RMQ(1,10) = ?



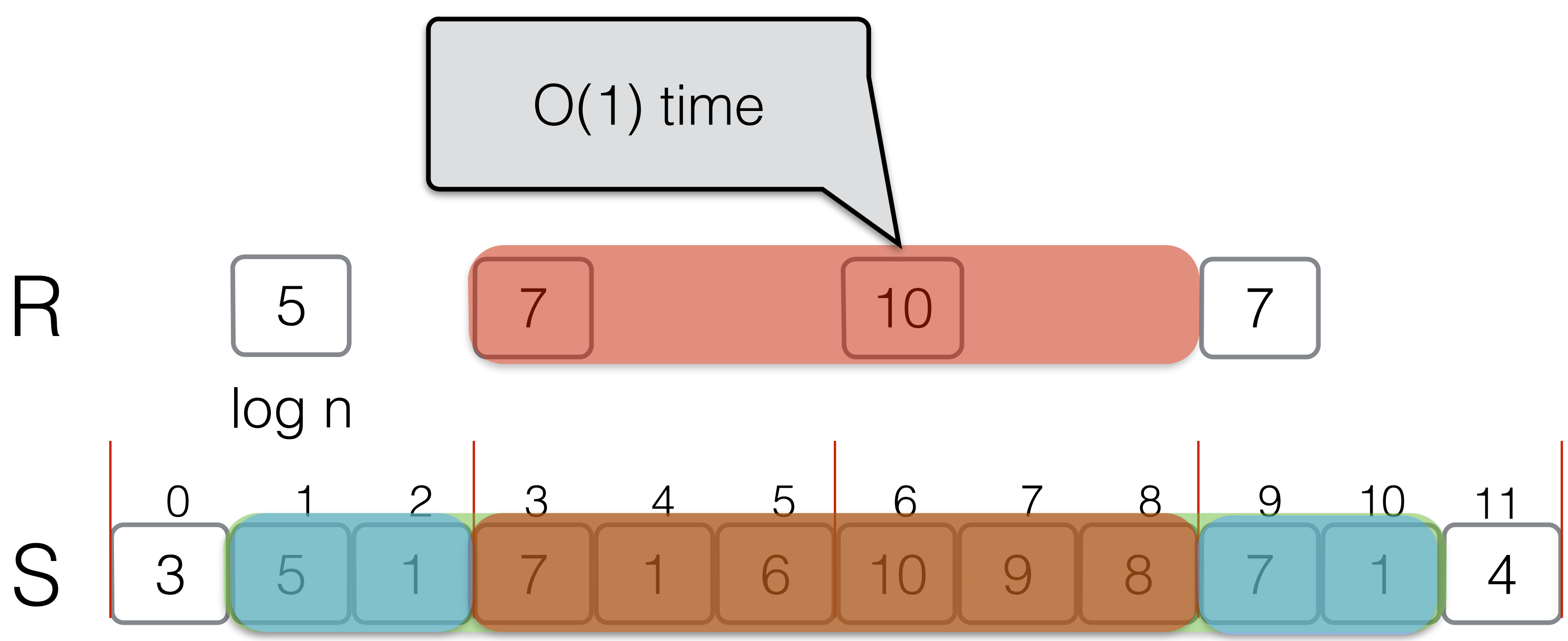
Range Maximum Query (3)

Space: O(n log n) bits
Query time: O(log n)

Use the previous solution on R!

Space: O(n log n) bits
Query time: O(1)

RMQ(1,10) = ?



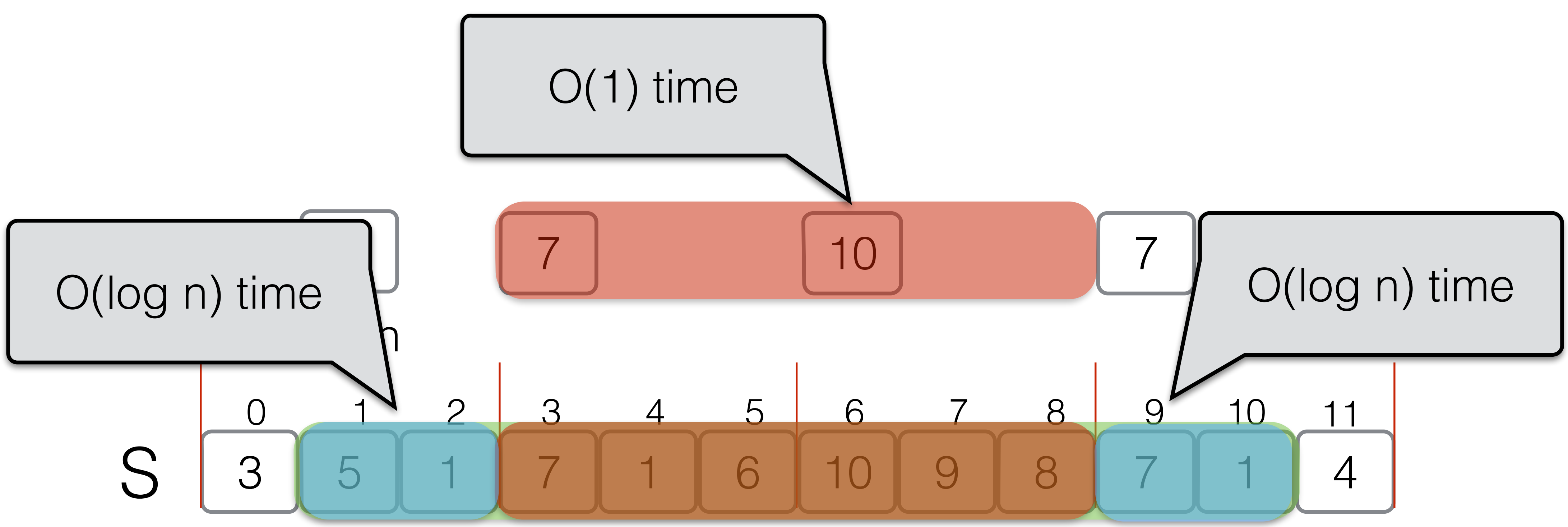
Range Maximum Query (3)

Space: O(n log n) bits
Query time: O(log n)

Use the previous solution on R!

Space: O(n log n) bits
Query time: O(1)

RMQ(1,10) = ?



Range Maximum Query (3)

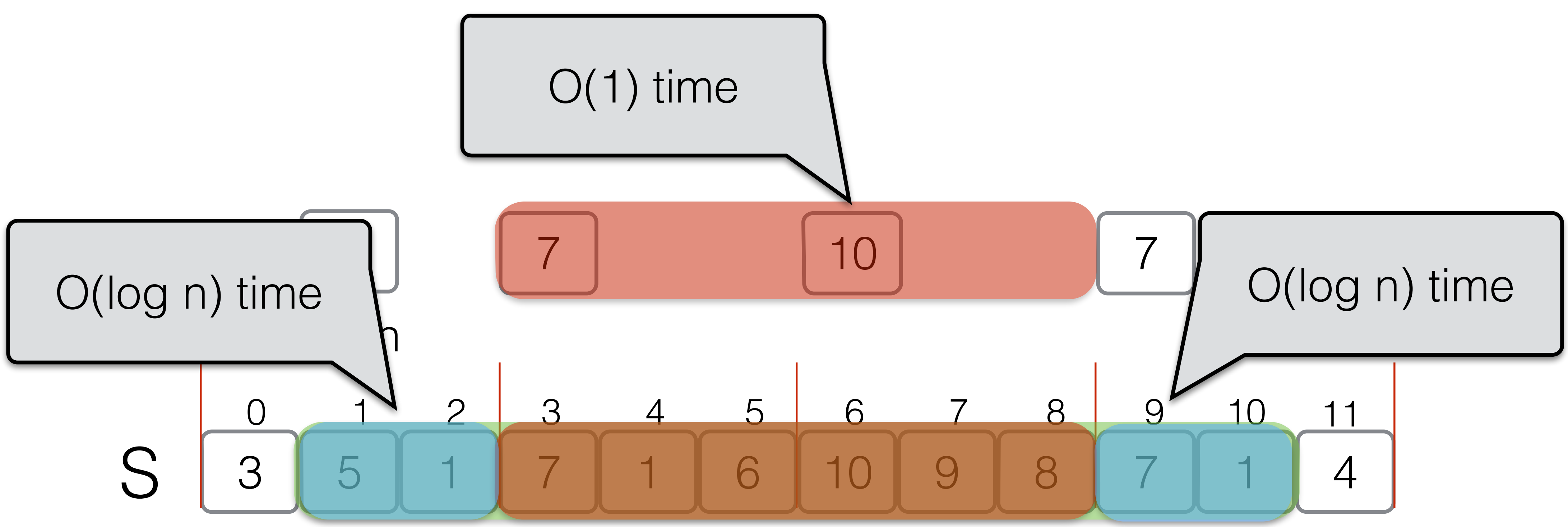
Space: O(n log n) bits
Query time: O(log n)

Space: O(n log n) bits
Query time: O(1)

Use the previous solution on R!

Space: O(n log n) bits
Query time: O(1)

RMQ(1,10) = ?



Range Maximum Query (3)

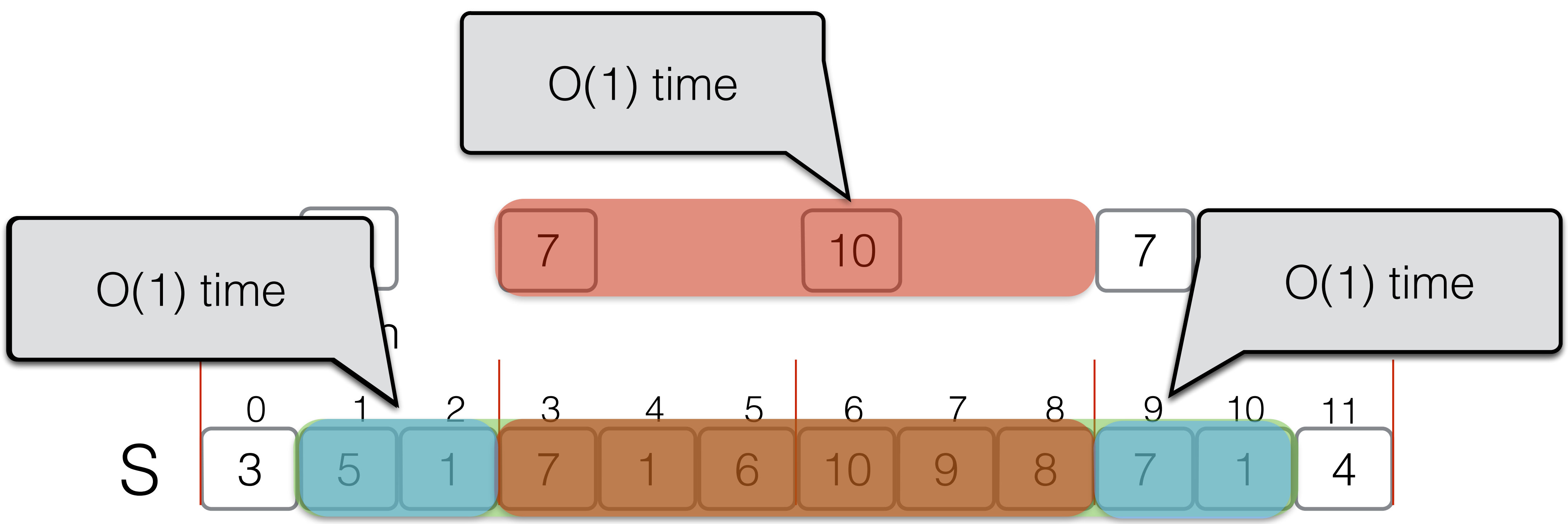
Space: O(n log n) bits
Query time: O(log n)

Space: O(n log n) bits
Query time: O(1)

Use the previous solution on R!

Space: O(n log n) bits
Query time: O(1)

RMQ(1,10) = ?



Range Maximum Query (3)

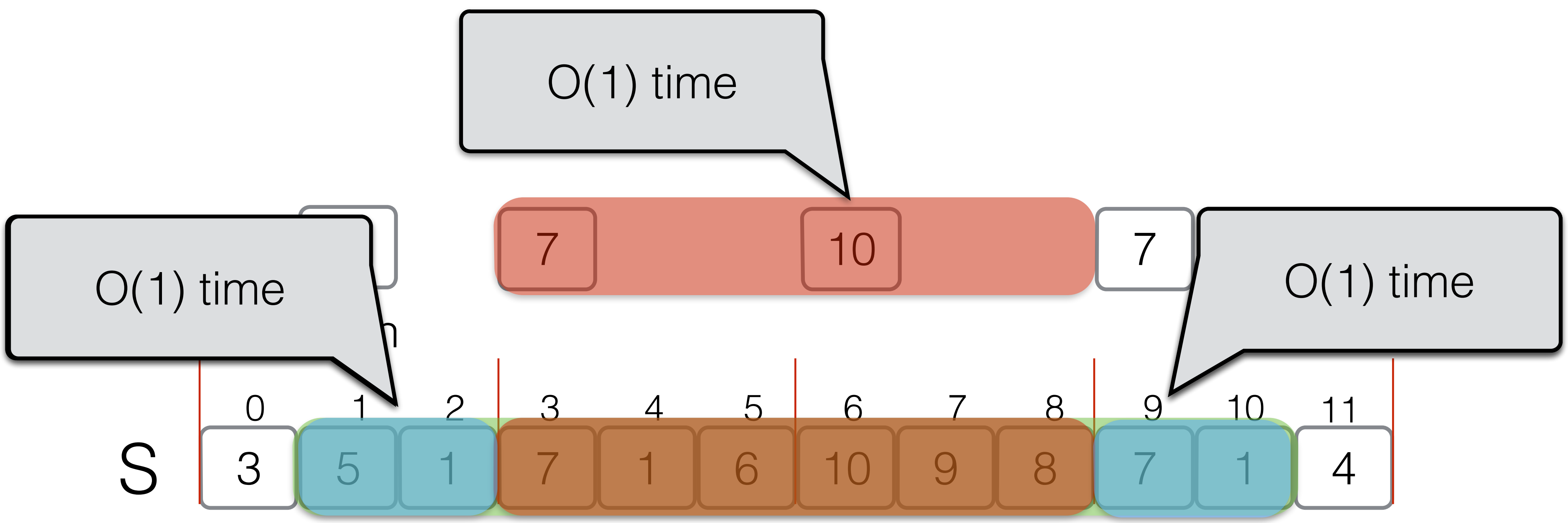
Space: O(n log n) bits
Query time: O(log n)

Space: O(n ~~log~~ n) bits
Query time: O(1)

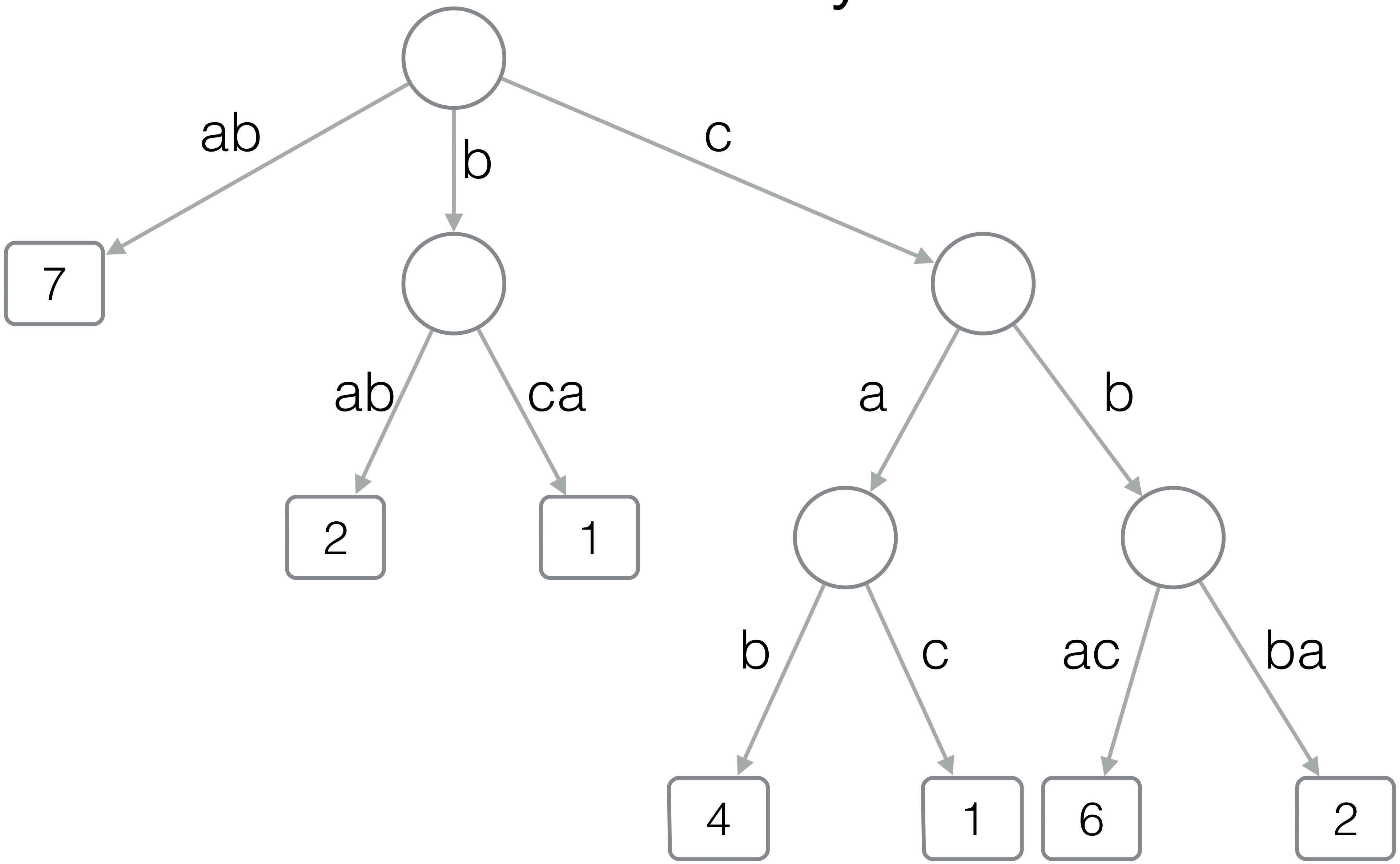
Use the previous solution on R!

Space: O(n log n) bits
Query time: O(1)

RMQ(1,10) = ?



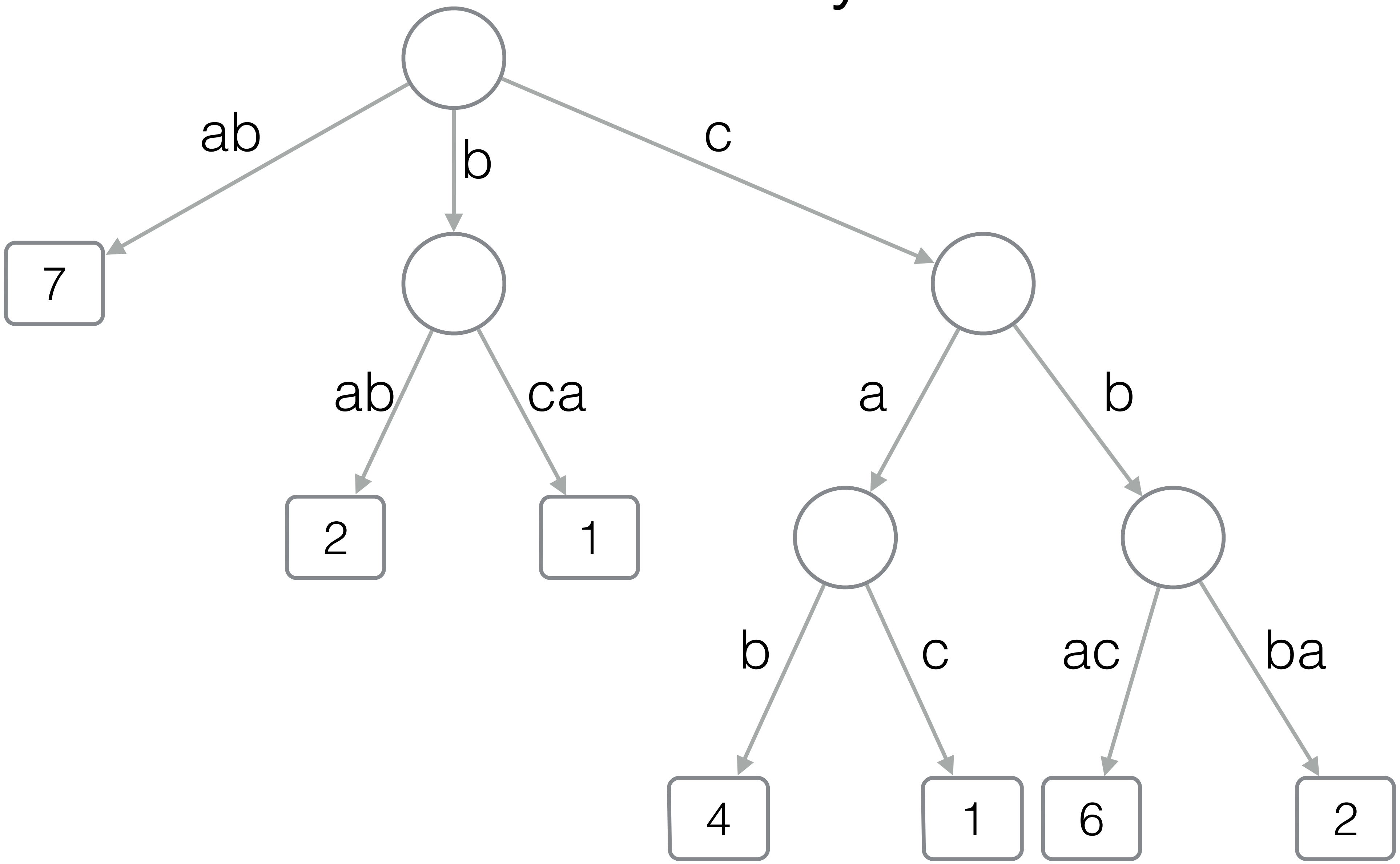
Summary



D = { ab (7), bab (2), bca (1), cab (4), cac (1), cbac (6), cbba (2) }

n = |D|, m total length of strings in D

Summary

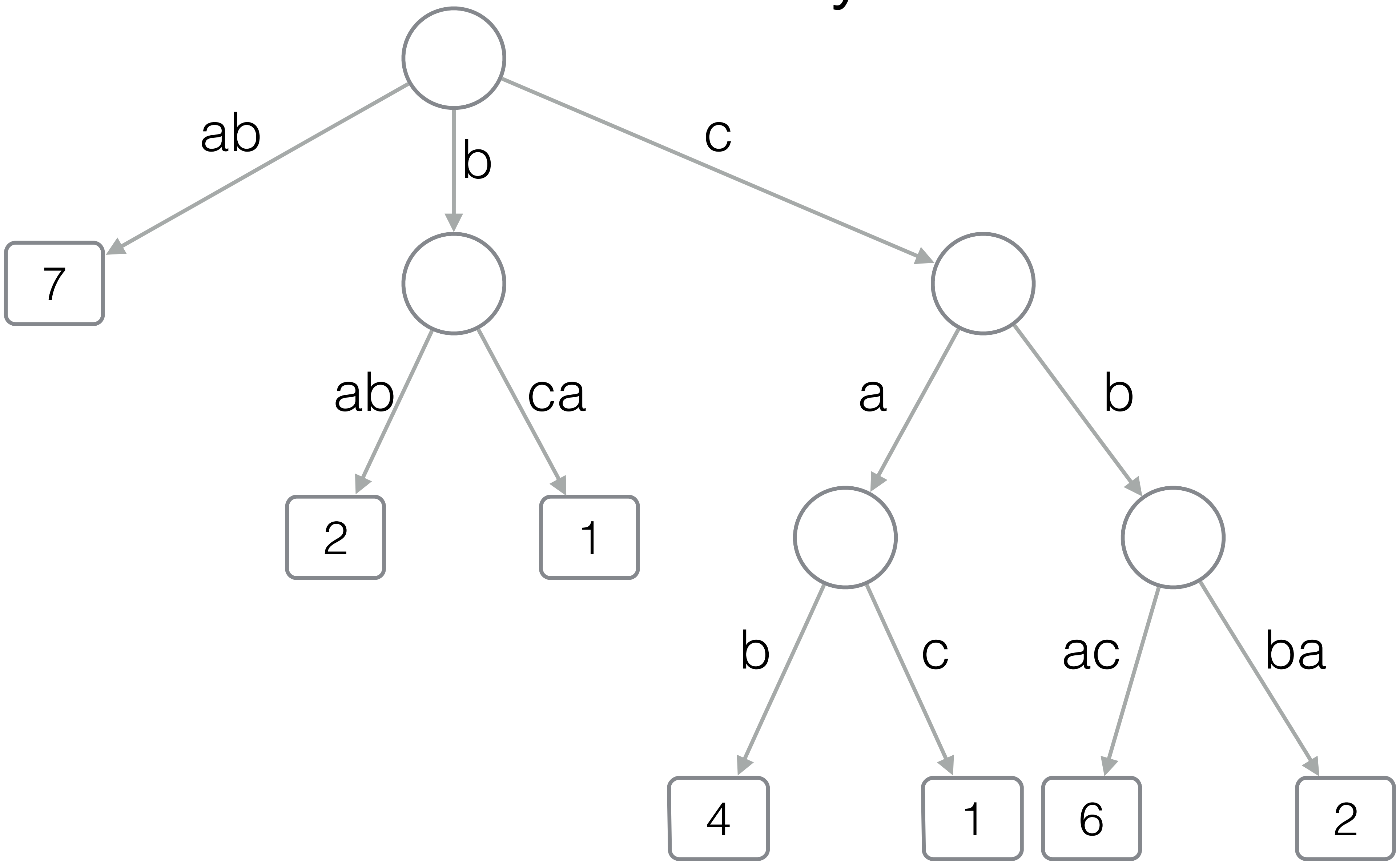


Find the node "prefixed" by P

D = { ab (7), bab (2), bca (1), cab (4), cac (1), cbac (6), cbba (2) }

n = |D|, m total length of strings in D

Summary



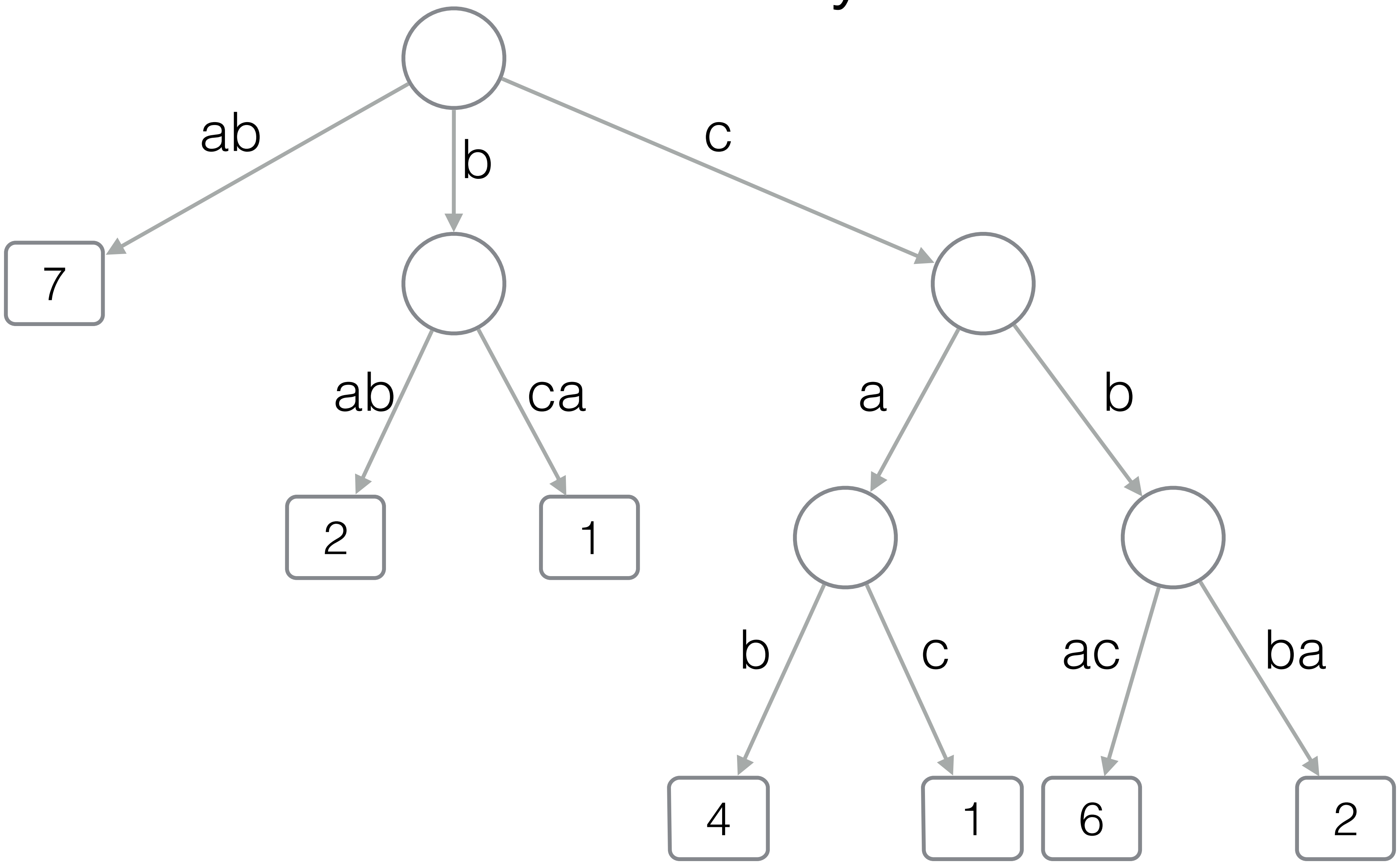
Find the node "prefixed" by P

O(|P|) time

D = { ab (7), bab (2), bca (1), cab (4), cac (1), cbac (6), cbba (2) }

n = |D|, m total length of strings in D

Summary

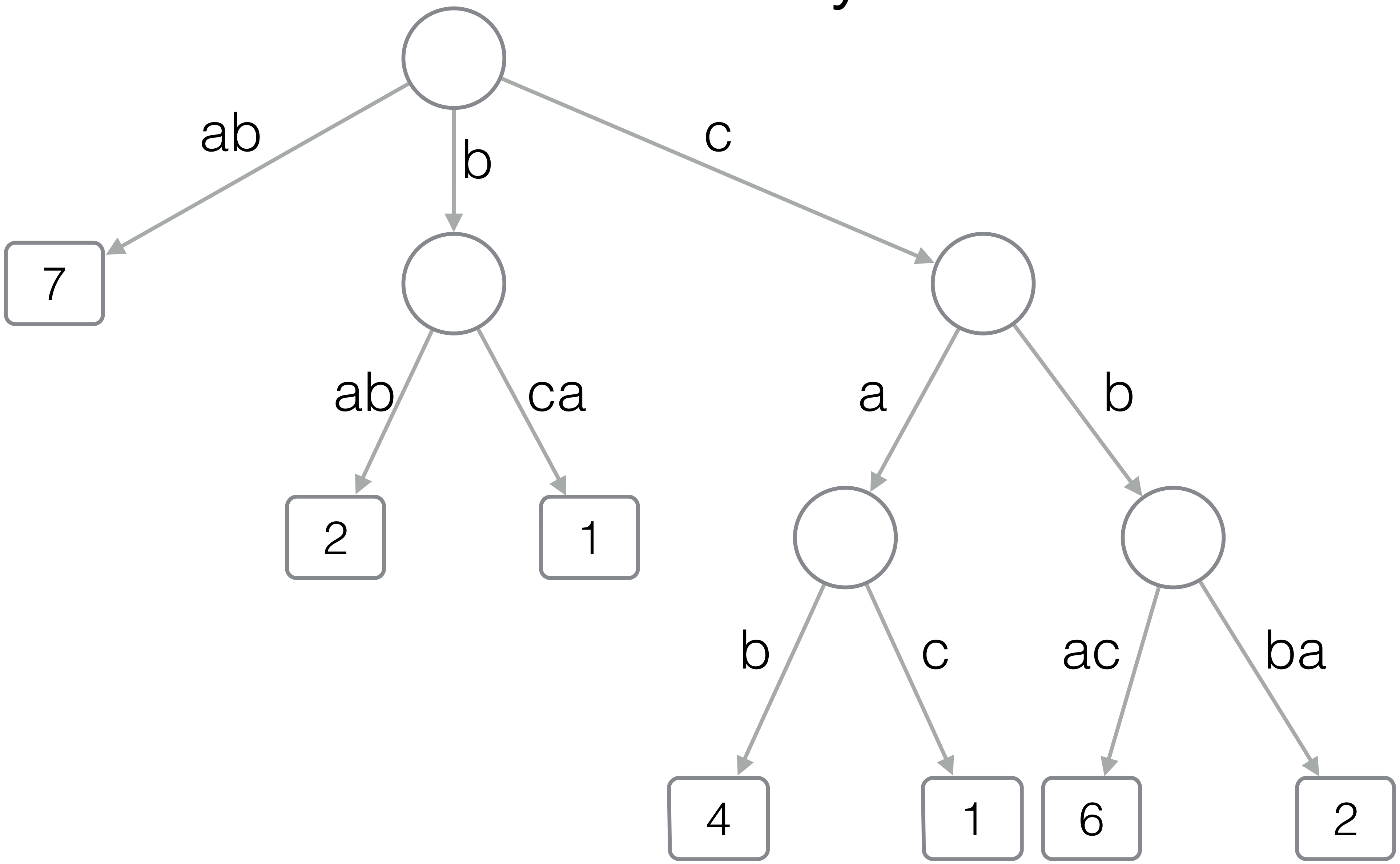


Find the node "prefixed" by P	O(|P|) time	O(m log σ + n log m) bits

D = { ab (7), bab (2), bca (1), cab (4), cac (1), cbac (6), cbba (2) }

n = |D|, m total length of strings in D

Summary



Find the node "prefixed" by P

O(|P|) time

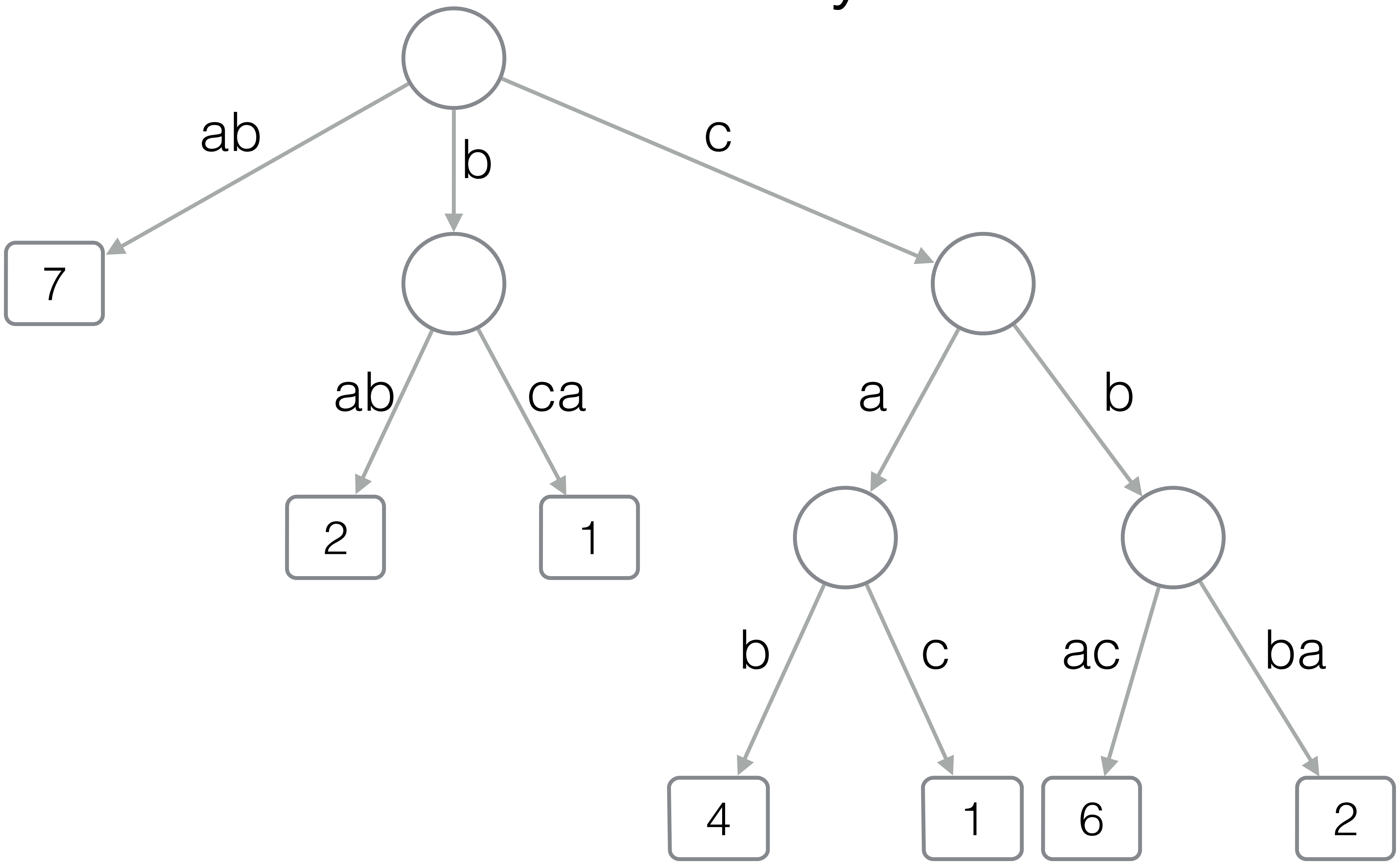
O(m log σ + n log m) bits

Compute the top-k strings

a (1), cab (4), cac (1), cbac (6), cbba (2) }

n = |D|, m total length of strings in D

Summary



Find the node “prefixed” by P

O(|P|) time

O(m log σ + n log m) bits

Compute the top-k strings

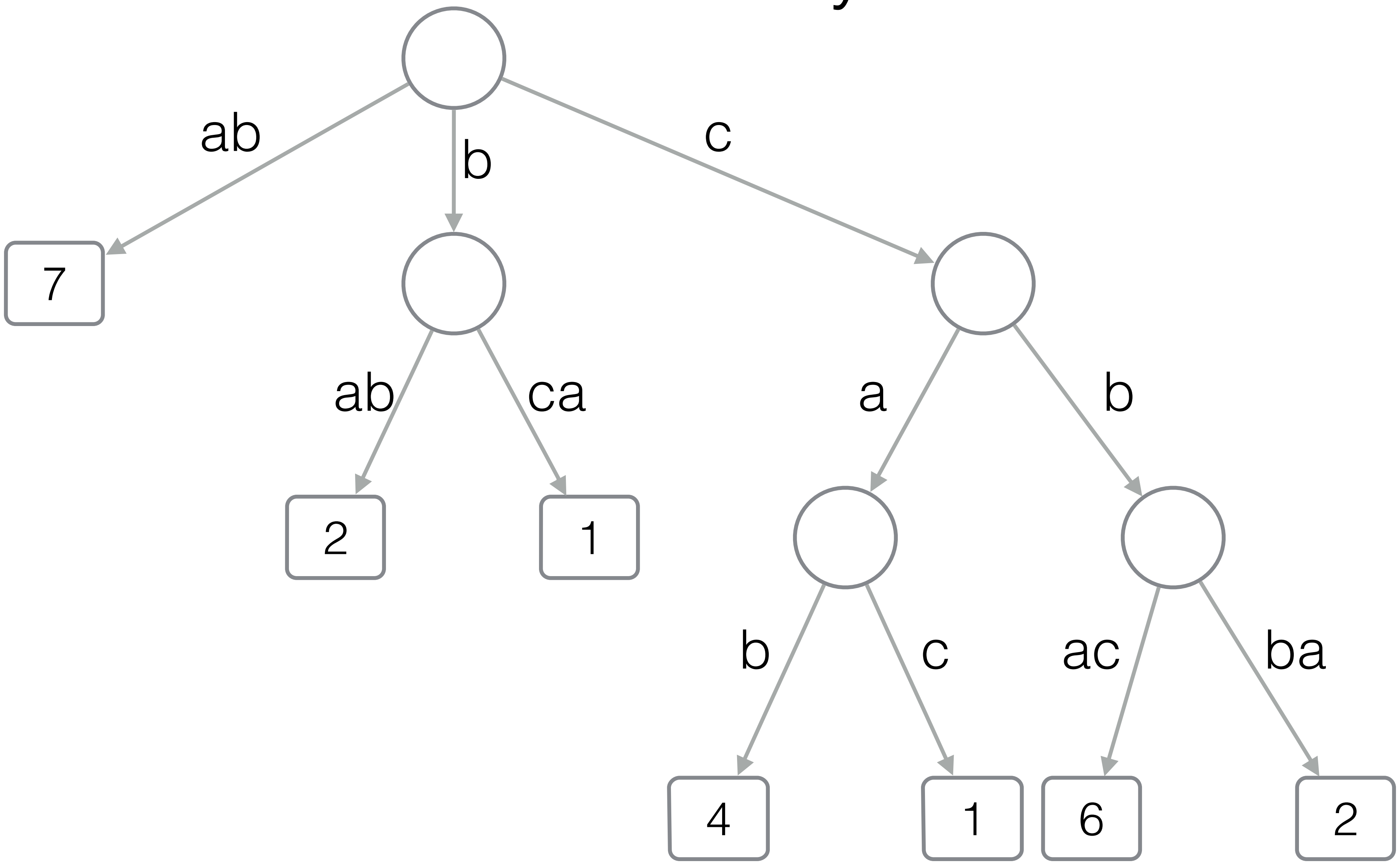
a

O(k log k) time

cbac (6), cbba (2) }

n = |D|, m total length of strings in D

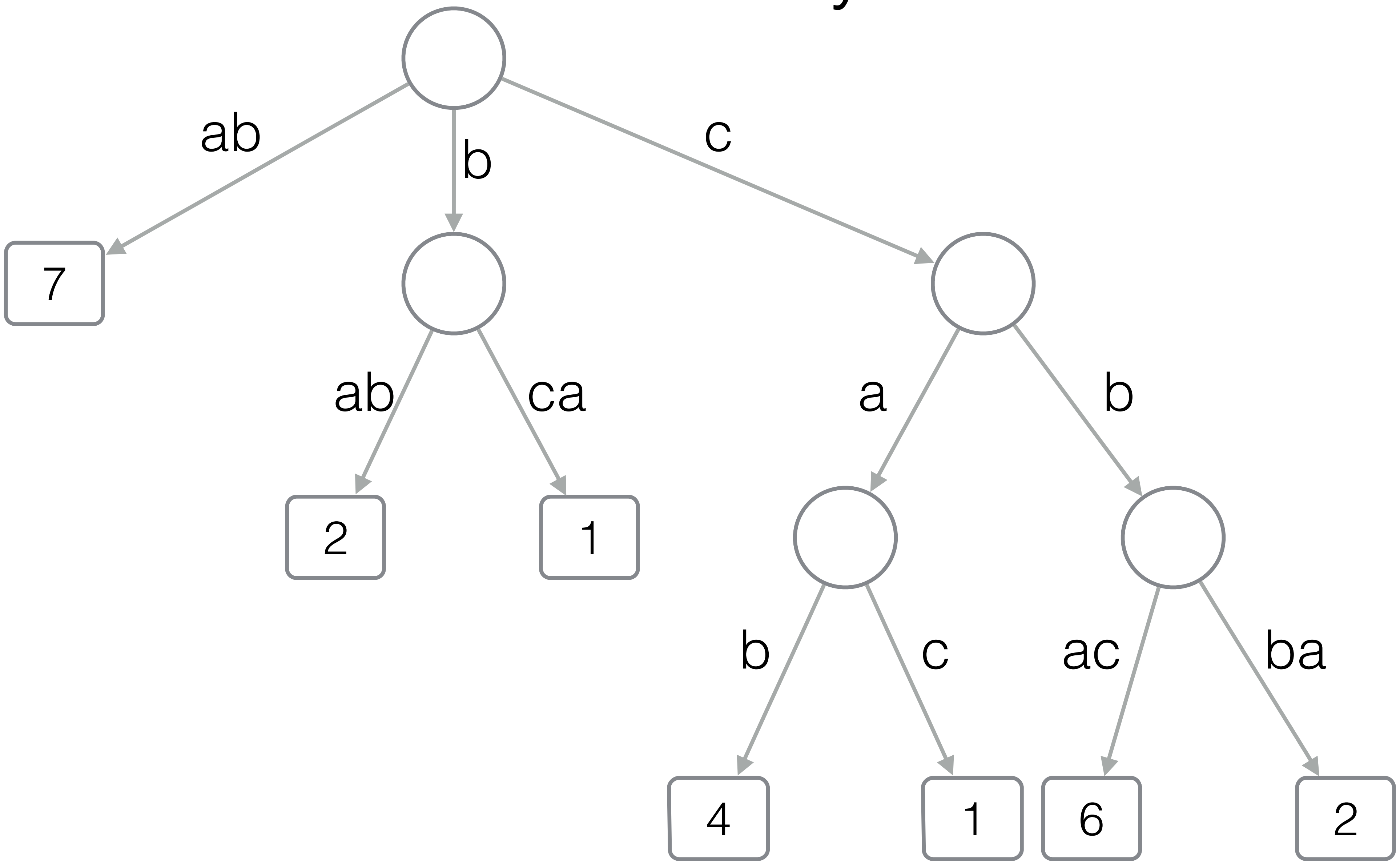
Summary



Find the node “prefixed” by P	O(|P|) time	O(m log σ + n log m) bits
Compute the top-k strings	O(k log k) time	O(n) bits

n = |D|, m total length of strings in D

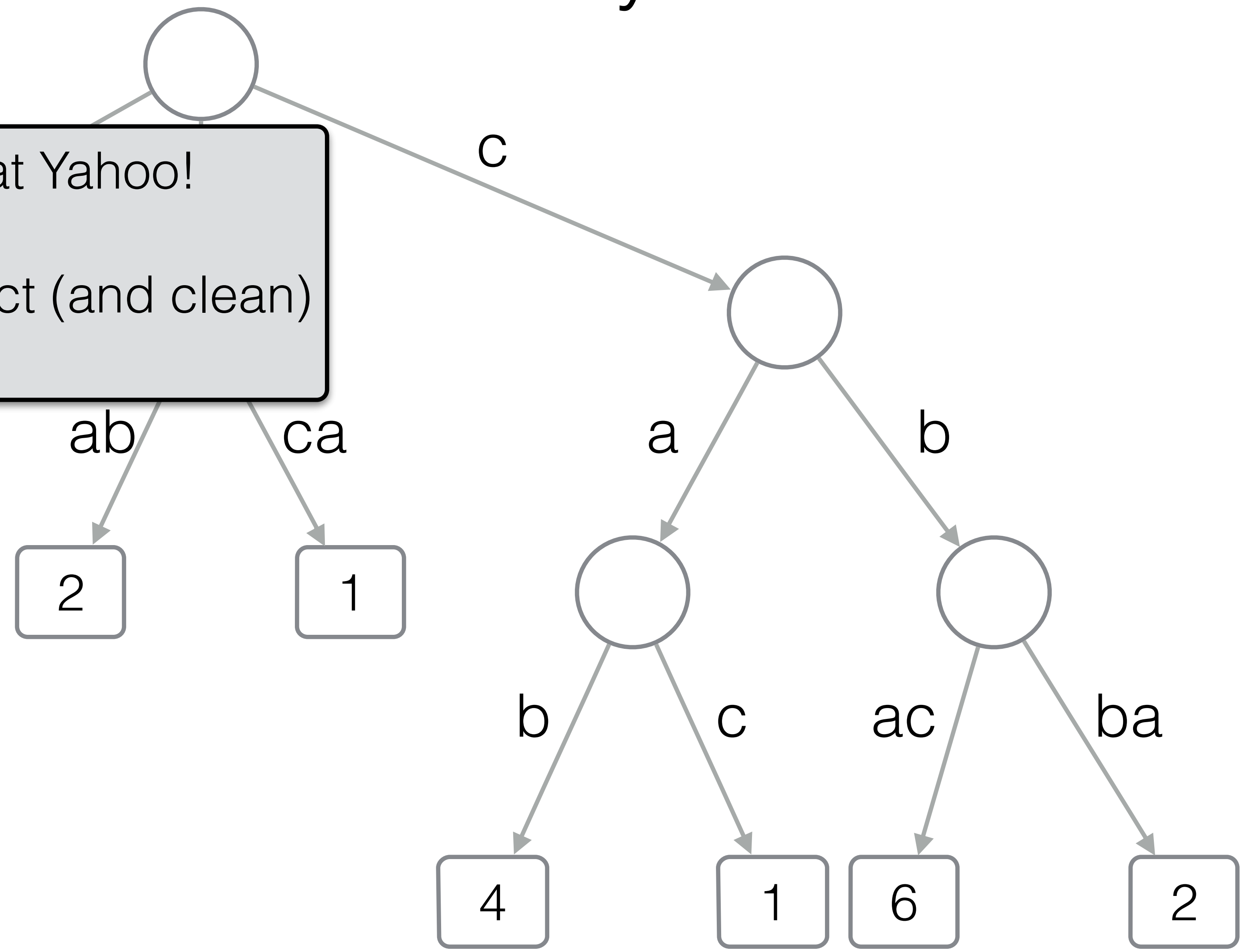
Summary



Find the node "prefixed" by P	O(|P|) time	O(m log σ + n log m) bits
Compute the top-k strings	O(k log k) time	O(n) bits

n = |D|, m total length of strings in D

Summary



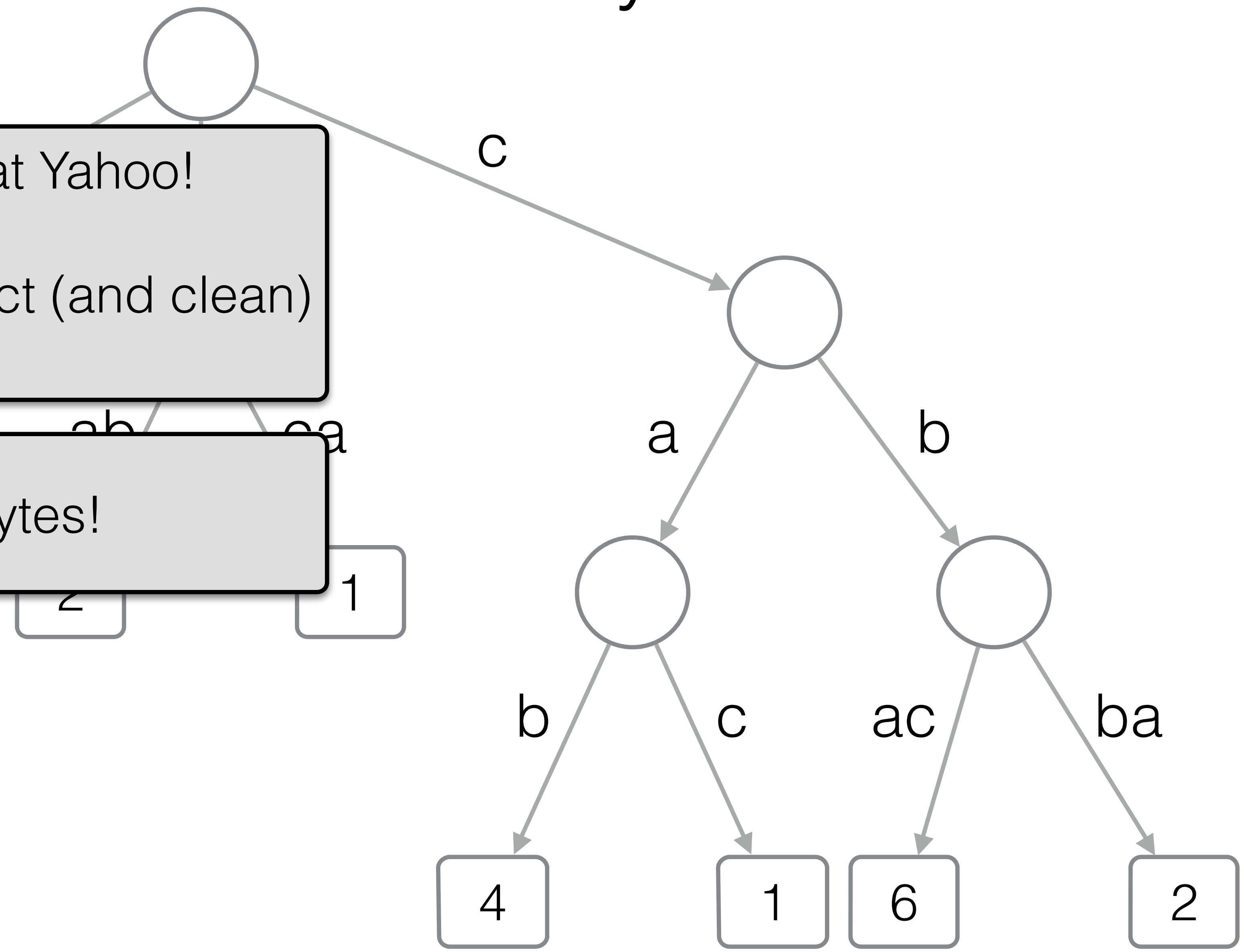
3 months query log at Yahoo!

≈600 million of distinct (and clean) queries

Find the node “prefixed” by P	O(|P|) time	O(m log σ + n log m) bits
Compute the top-k strings	O(k log k) time	O(n) bits

n = \|D\|, m total length of strings in D

Summary



3 months query log at Yahoo!

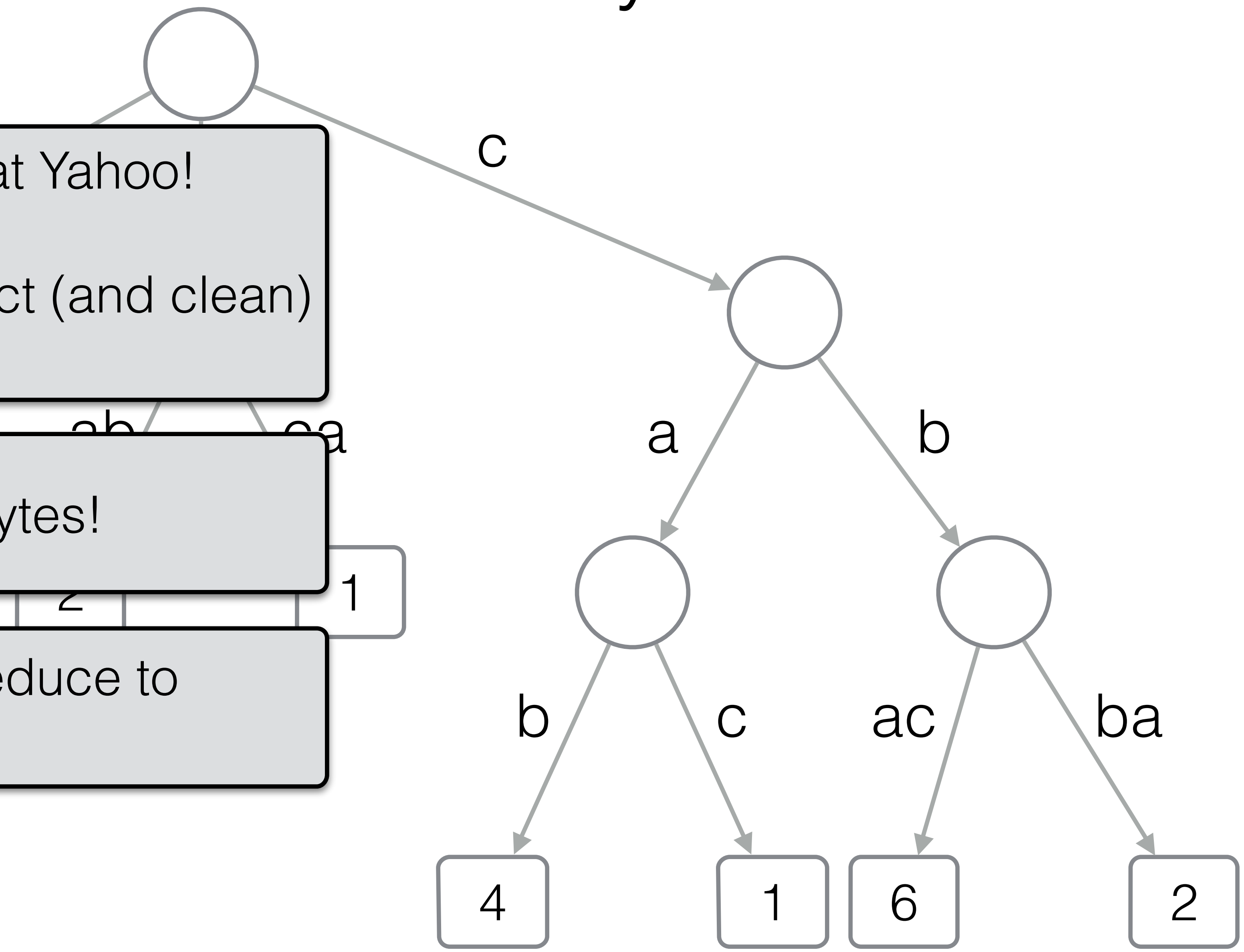
≈600 million of distinct (and clean) queries

Trie requires ≈50 Gbytes!

Find the node "prefixed" by P	O(|P|) time	O(m log σ + n log m) bits
Compute the top-k strings	O(k log k) time	O(n) bits

n = |D|, m total length of strings in D

Summary



3 months query log at Yahoo!

≈600 million of distinct (and clean) queries

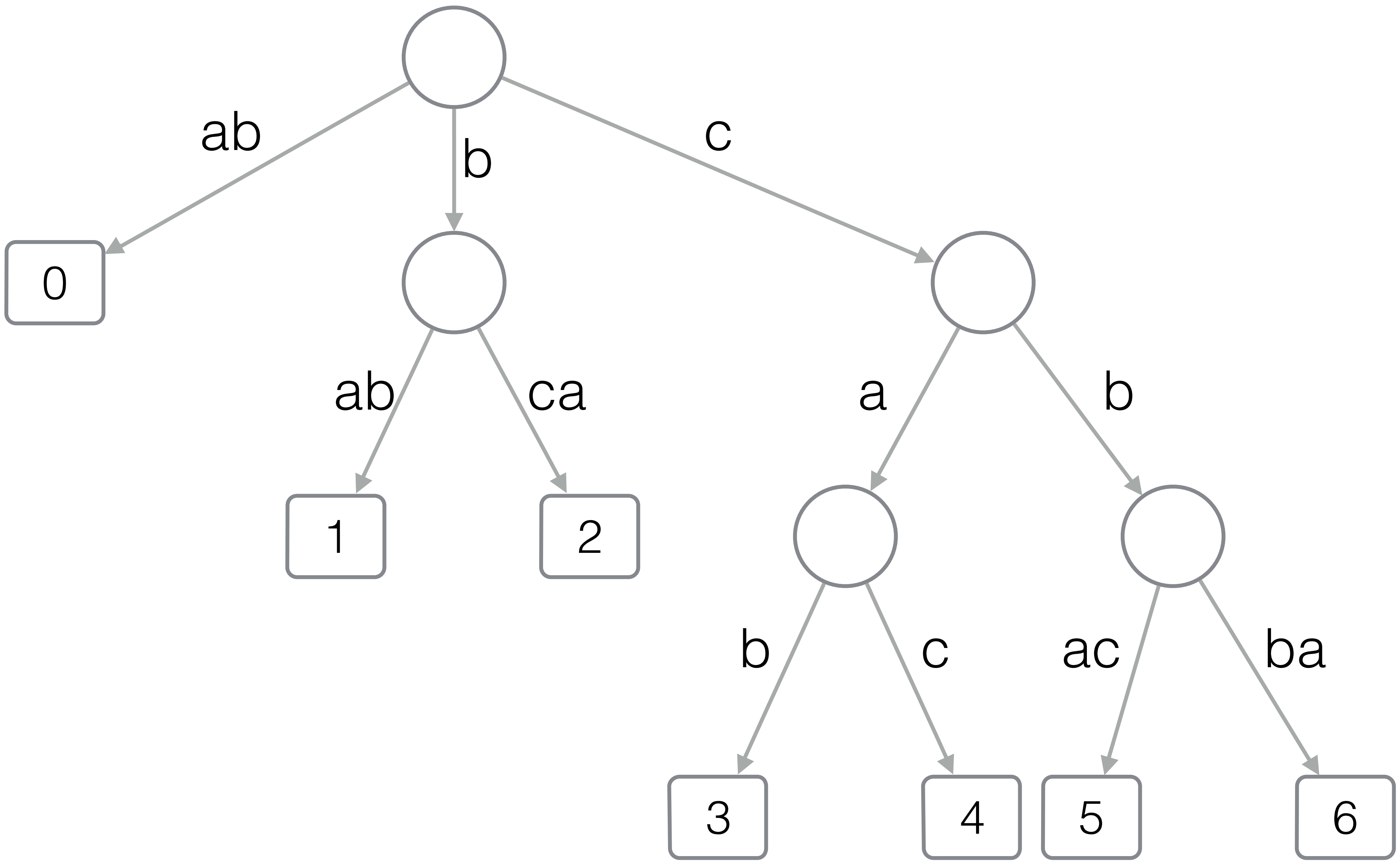
Trie requires ≈50 Gbytes!

We will see how to reduce to ≈5 Gbytes!

Find the node "prefixed" by P	O(|P|) time	O(m log σ + n log m) bits
Compute the top-k strings	O(k log k) time	O(n) bits

n = |D|, m total length of strings in D

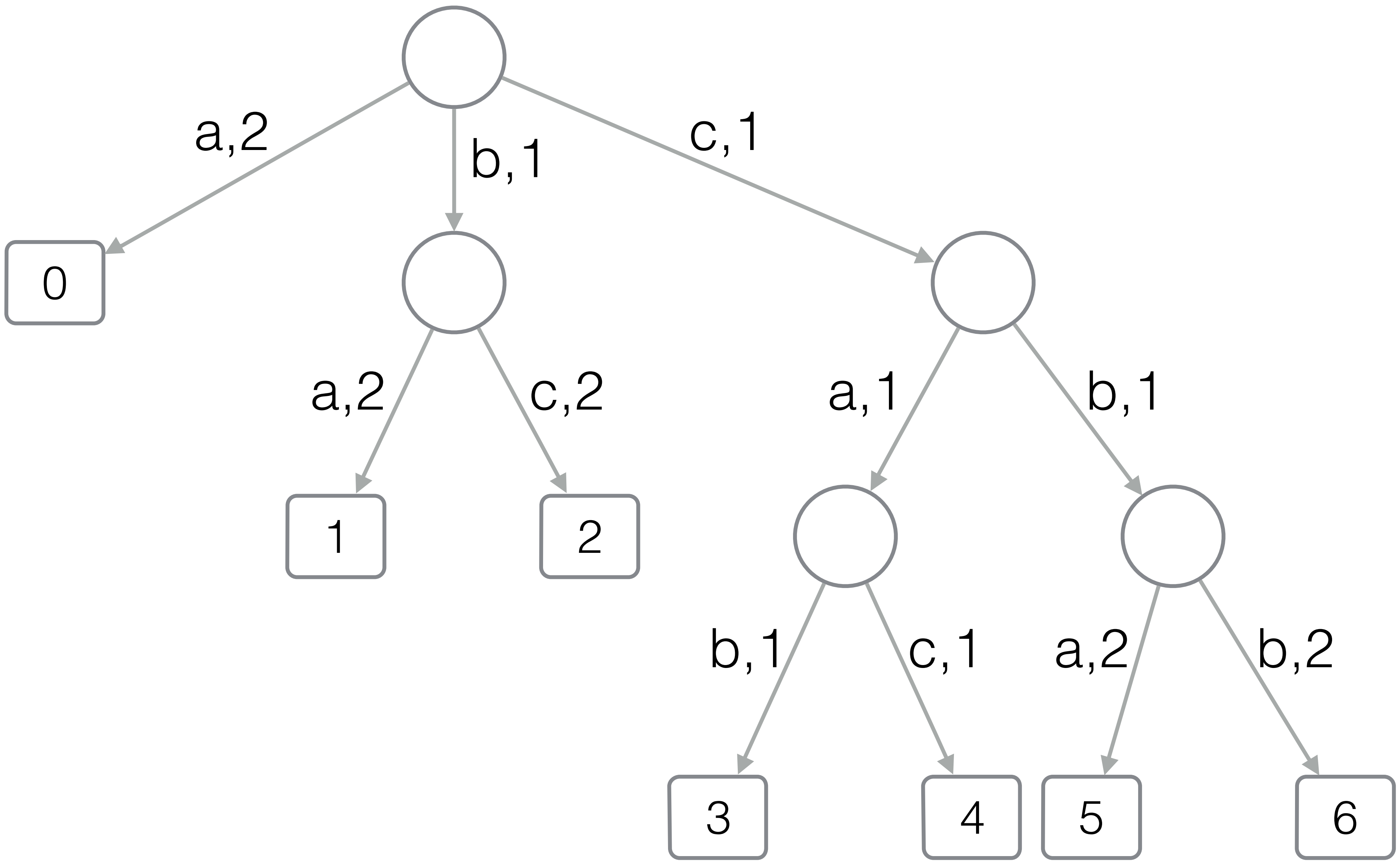
Patricia trie



D = { ab, bab, bca, cab, cac, cbac, cbba }

n = |D|, m total length of strings in D

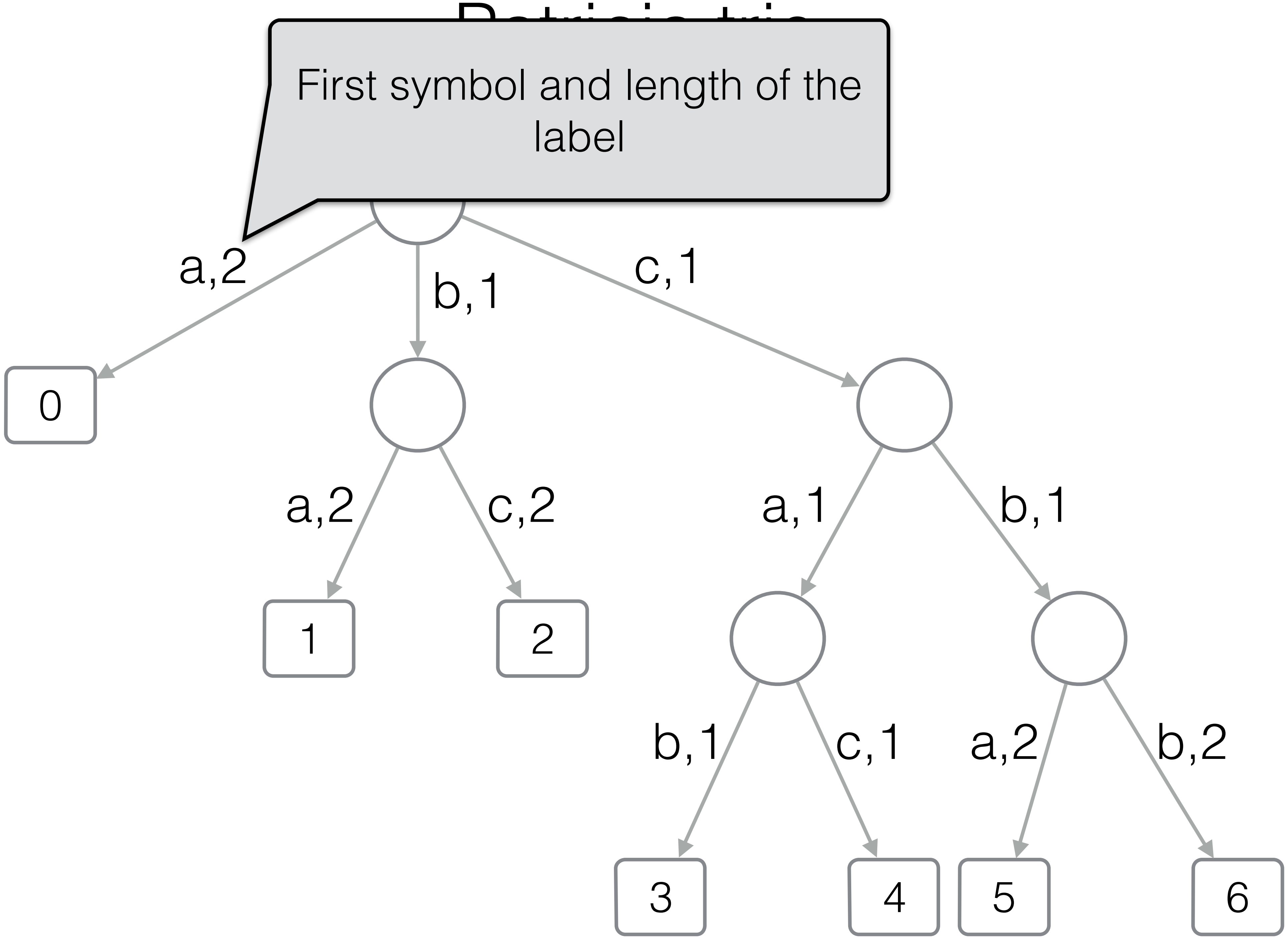
Patricia trie



D = { ab, bab, bca, cab, cac, cbac, cbba }

n = |D|, m total length of strings in D

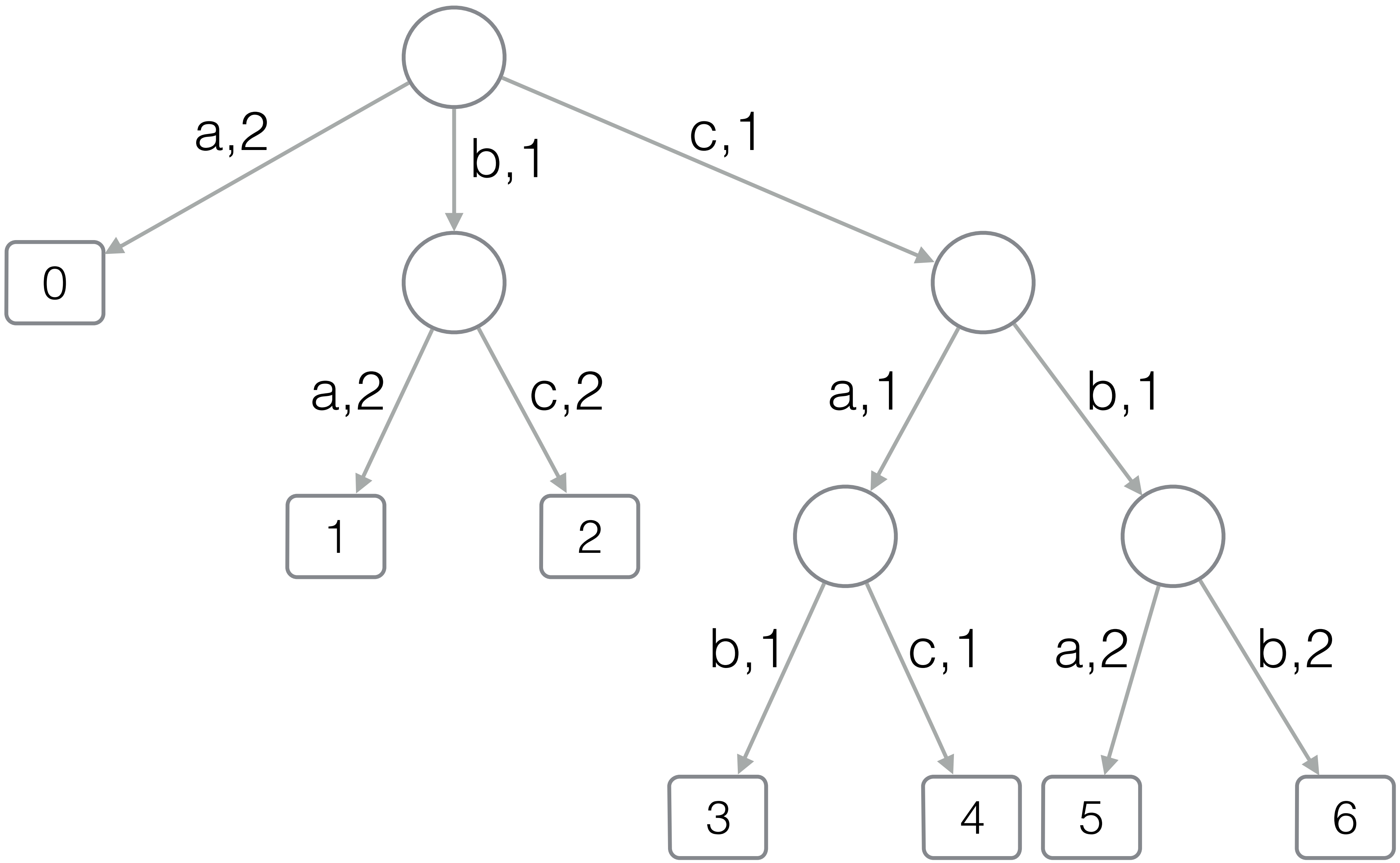
Patricia trie



D = { ab, bab, bca, cab, cac, cbac, cbba }

n = |D|, m total length of strings in D

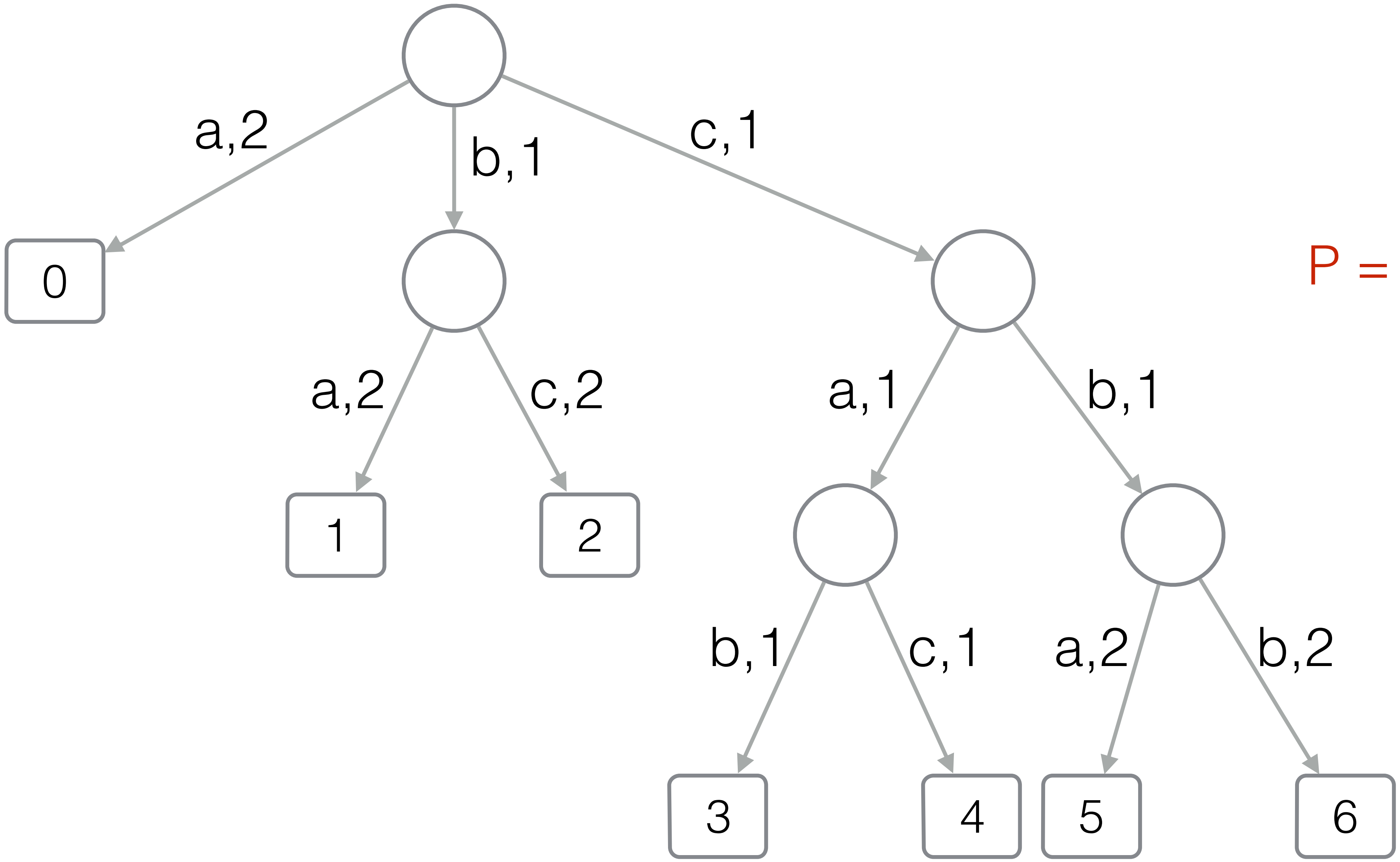
Patricia trie



D = { ab, bab, bca, cab, cac, cbac, cbba }

n = |D|, m total length of strings in D

Patricia trie

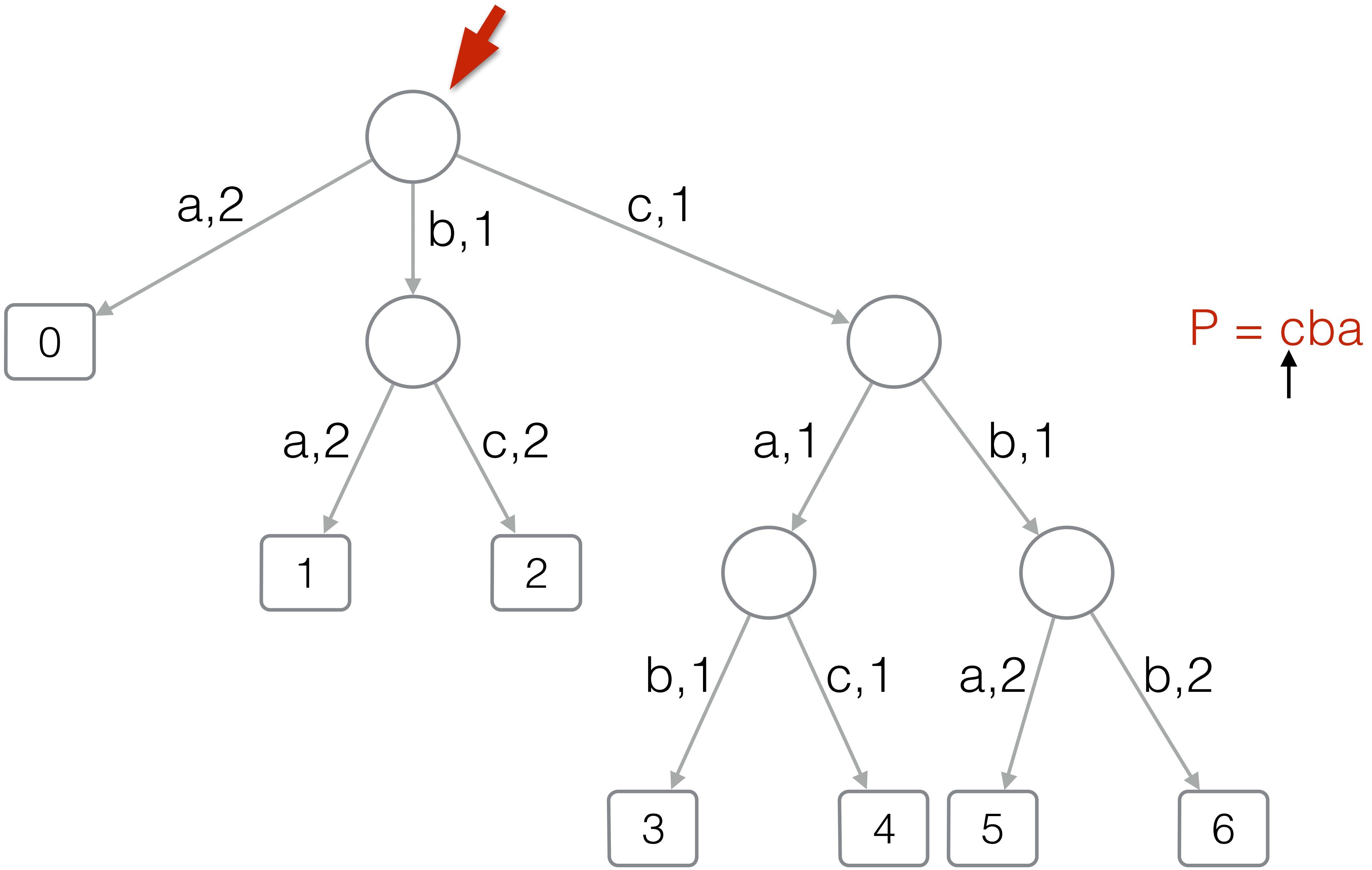


P = cba

D = { ab, bab, bca, cab, cac, cbac, cbba }

n = |D|, m total length of strings in D

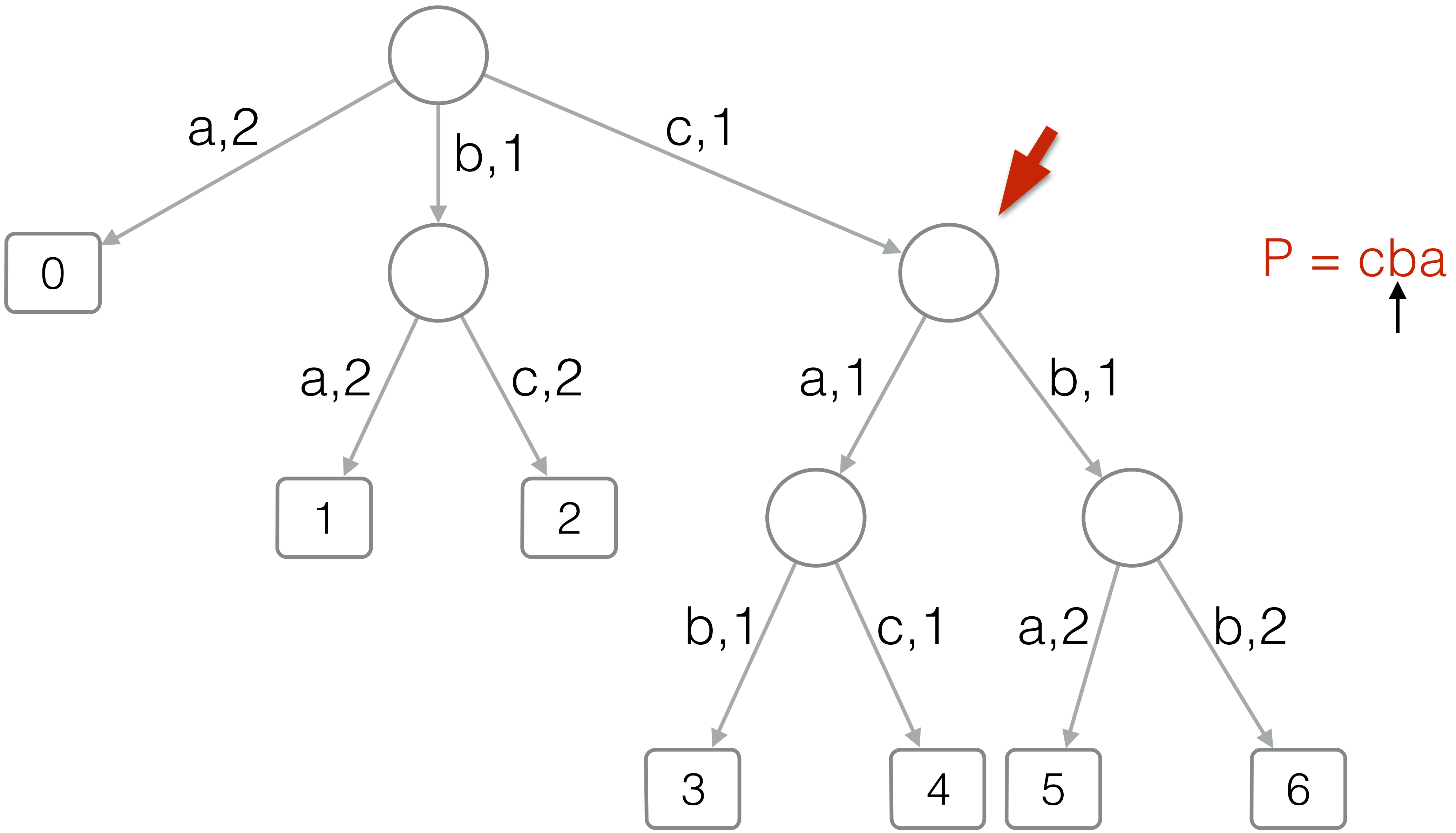
Patricia trie



D = { ab, bab, bca, cab, cac, cbac, cbba }

n = |D|, m total length of strings in D

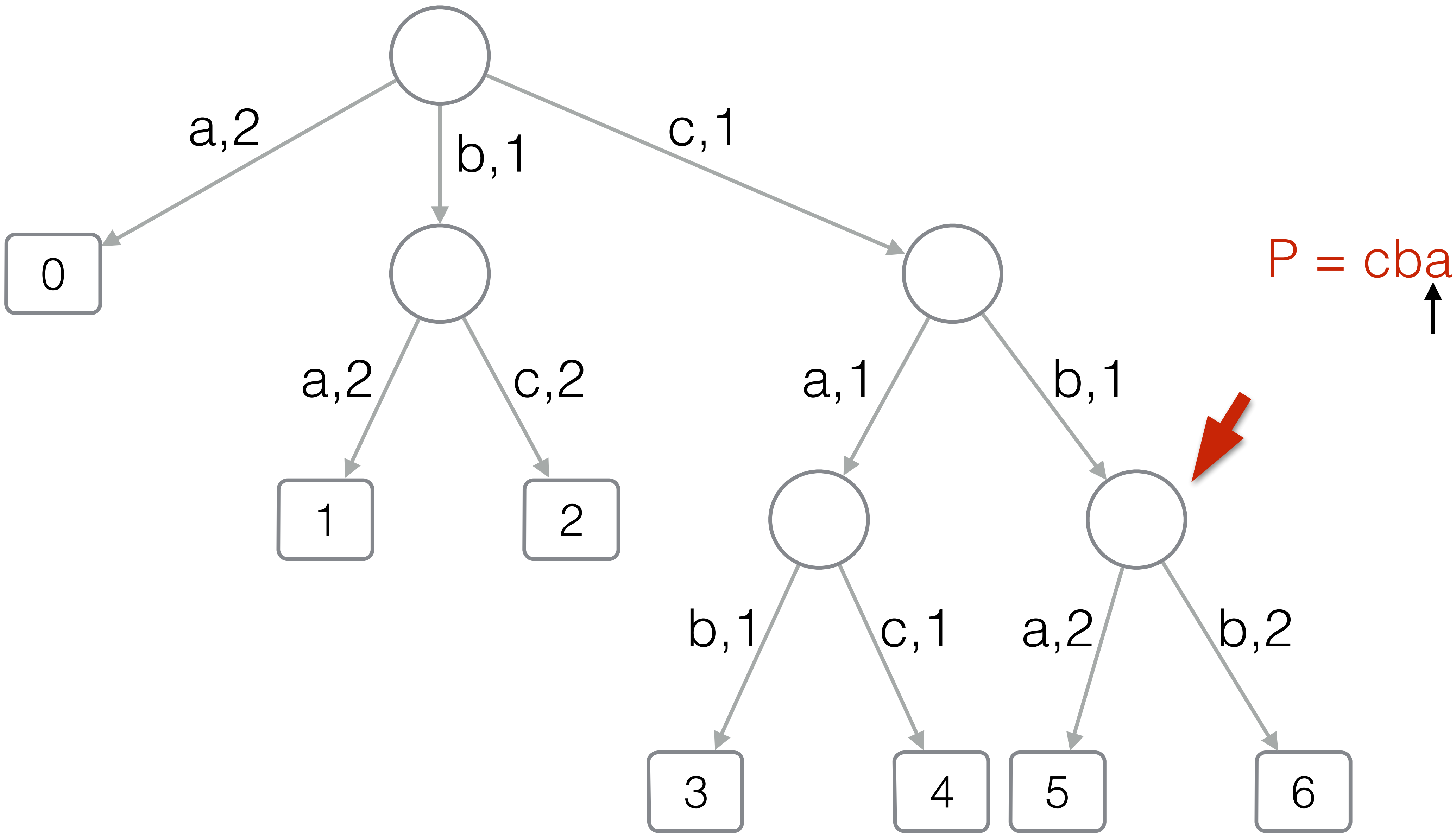
Patricia trie



D = { ab, bab, bca, cab, cac, cbac, cbba }

n = |D|, m total length of strings in D

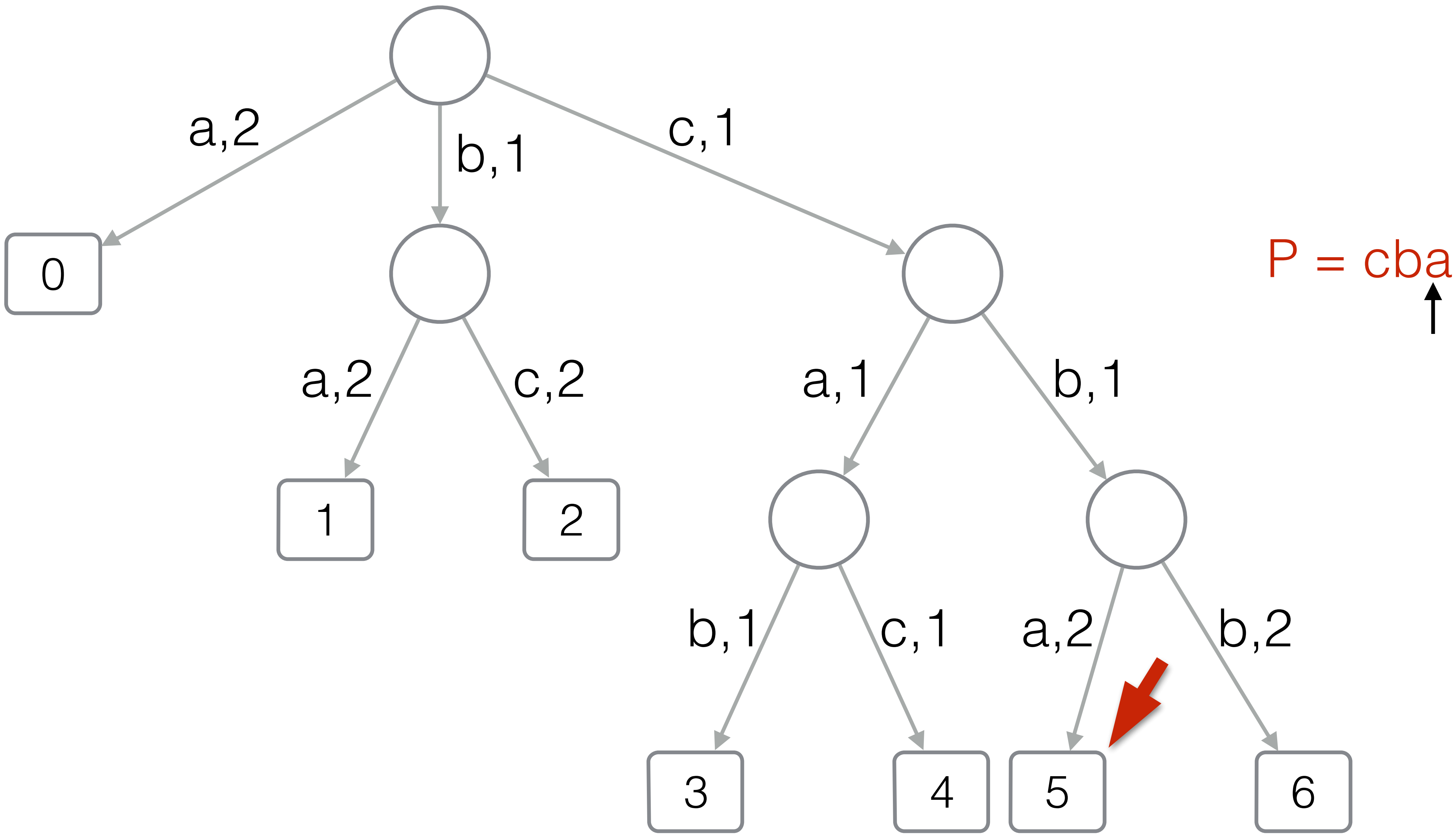
Patricia trie



D = { ab, bab, bca, cab, cac, cbac, cbba }

n = |D|, m total length of strings in D

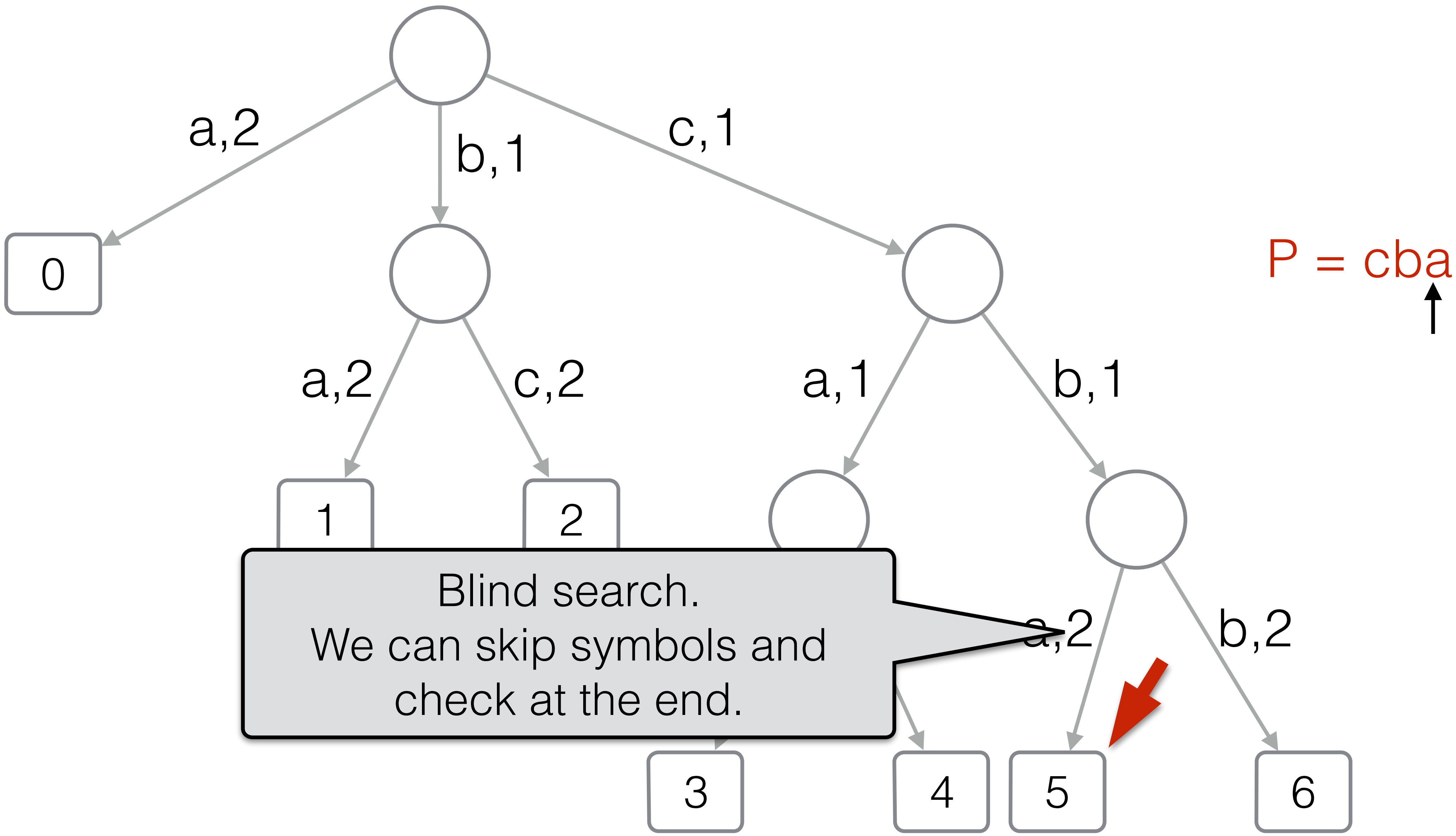
Patricia trie



D = { ab, bab, bca, cab, cac, cbac, cbba }

n = |D|, m total length of strings in D

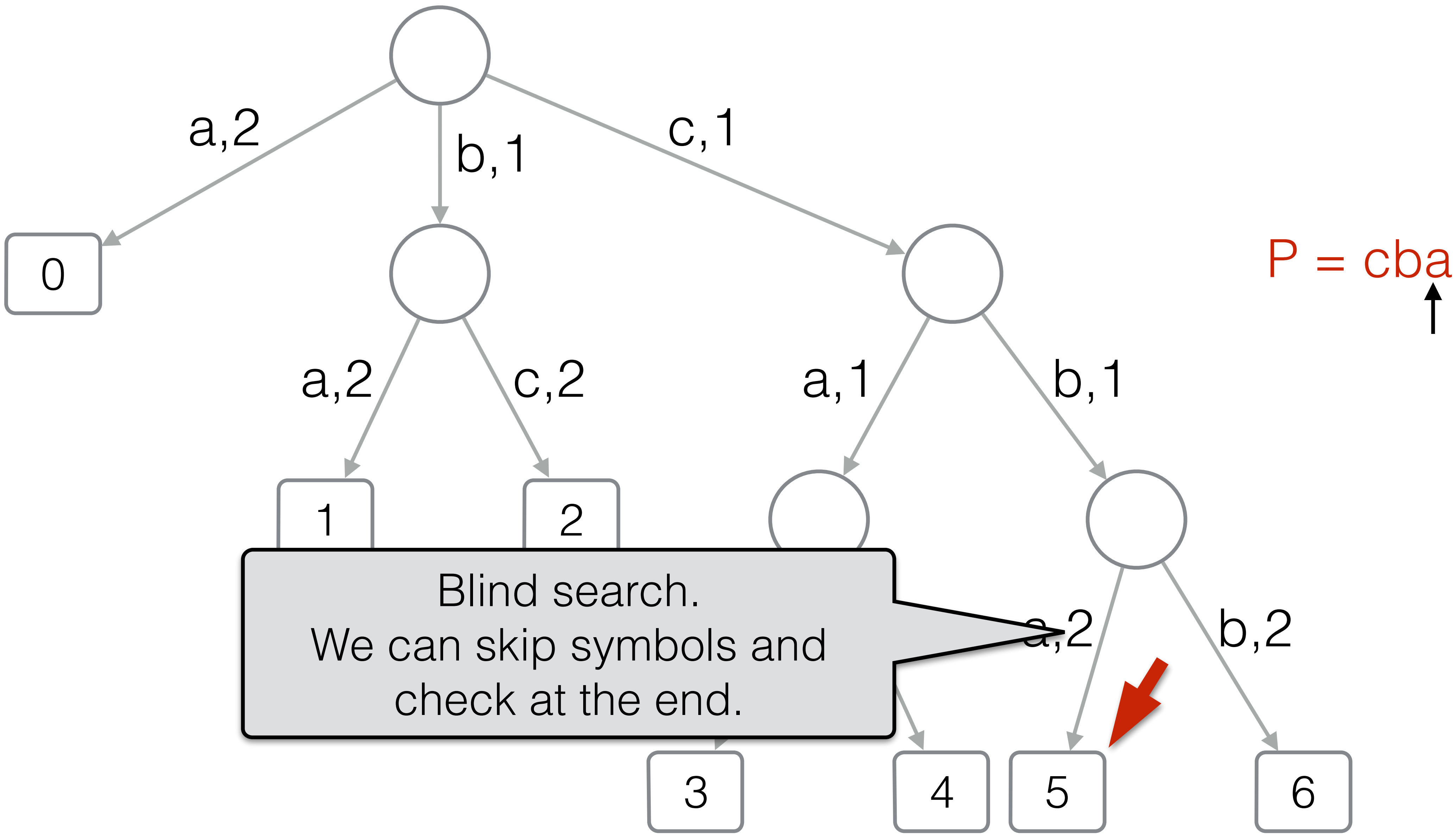
Patricia trie



D = { ab, bab, bca, cab, cac, cbac, cbba }

n = |D|, m total length of strings in D

Patricia trie



O(|P|) time

O(n log m + m log σ) bits

n = |D|, m total length of strings in D

Rank/Select queries

	0	1	2	3	4	5	6	7	8	9	10	11
B	0	0	1	1	1	0	1	0	1	1	0	1

Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

	0	1	2	3	4	5	6	7	8	9	10	11
B	0	0	1	1	1	0	1	0	1	1	0	1

Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

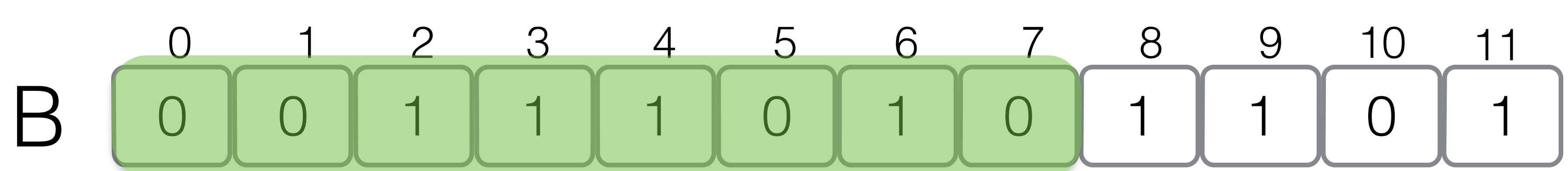
$Rank_0(7)$ =

	0	1	2	3	4	5	6	7	8	9	10	11
B	0	0	1	1	1	0	1	0	1	1	0	1

Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_0(7) = 4$

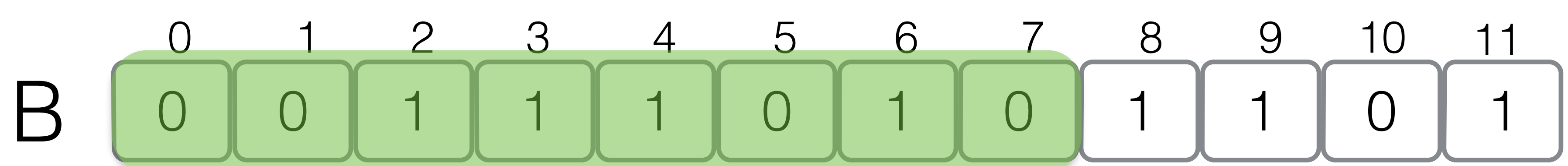


Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_1(j)$ = # of 1 in B[0,j]

$Rank_0(7) = 4$



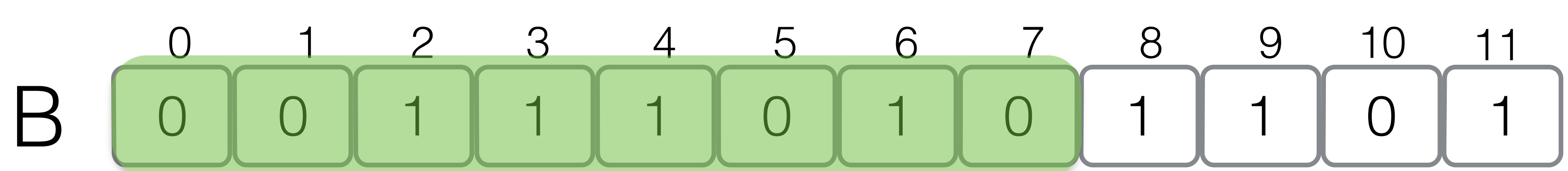
Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_1(j)$ = # of 1 in B[0,j]

$Rank_0(7) = 4$

$Rank_1(7) = 8 - Rank_0(7) = 4$



Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

$Select_0(j)$ = position of the j-th 0 in B

$Rank_1(j)$ = # of 1 in B[0,j]

$Rank_0(7) = 4$

$Rank_1(7) = 8 - Rank_0(7) = 4$

	0	1	2	3	4	5	6	7	8	9	10	11
B	0	0	1	1	1	0	1	0	1	1	0	1

Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_1(j)$ = # of 1 in B[0,j]

$Select_0(j)$ = position of the j-th 0 in B

$Select_1(j)$ = position of the j-th 0 in B

$Rank_0(7) = 4$

$Rank_1(7) = 8 - Rank_0(7) = 4$

	0	1	2	3	4	5	6	7	8	9	10	11
B	0	0	1	1	1	0	1	0	1	1	0	1

Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_1(j)$ = # of 1 in B[0,j]

$Rank_0(7) = 4$

$Rank_1(7) = 8 - Rank_0(7) = 4$

$Select_0(j)$ = position of the j-th 0 in B

$Select_1(j)$ = position of the j-th 0 in B

$Select_1(4) =$

	0	1	2	3	4	5	6	7	8	9	10	11
B	0	0	1	1	1	0	1	0	1	1	0	1

Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_1(j)$ = # of 1 in B[0,j]

$Rank_0(7) = 4$

$Rank_1(7) = 8 - Rank_0(7) = 4$

$Select_0(j)$ = position of the j-th 0 in B

$Select_1(j)$ = position of the j-th 0 in B

$Select_1(4) = 6$

	0	1	2	3	4	5	6	7	8	9	10	11
B	0	0	1	1	1	0	1	0	1	1	0	1

Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_1(j)$ = # of 1 in B[0,j]

$Select_0(j)$ = position of the j-th 0 in B

$Select_1(j)$ = position of the j-th 0 in B

(4) = 6

Space: n + O(n log log n/log n) bits
Query time: O(1)

	0	1	2	3	4	5	6	7	8	9	10	11
B	0	0	1	1	1	0	1	0	1	1	0	1

Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_1(j)$ = # of 1 in B[0,j]

$Select_0(j)$ = position of the j-th 0 in B

$Select_1(j)$ = position of the j-th 0 in B

(4) = 6

Space: n + O(n log log n/log n) bits
Query time: O(1)

	0	1	2	3	4	5	6	7	8	9	10	11
B	0	0	1	1	1	0	1	0	1	1	0	1

Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_1(j)$ = # of 1 in B[0,j]

Space: n + O(n log log n/log n) bits
Query time: O(1)

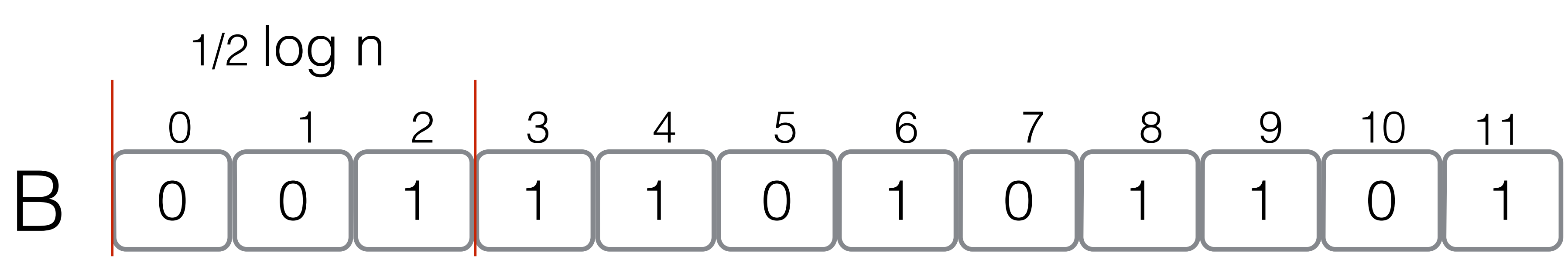
	0	1	2	3	4	5	6	7	8	9	10	11
B	0	0	1	1	1	0	1	0	1	1	0	1

Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_1(j)$ = # of 1 in B[0,j]

Space: n + O(n log log n/log n) bits
Query time: O(1)



Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_1(j)$ = # of 1 in B[0,j]

> Space: n + O(n log log n/log n) bits
> Query time: O(1)

1/2 log n

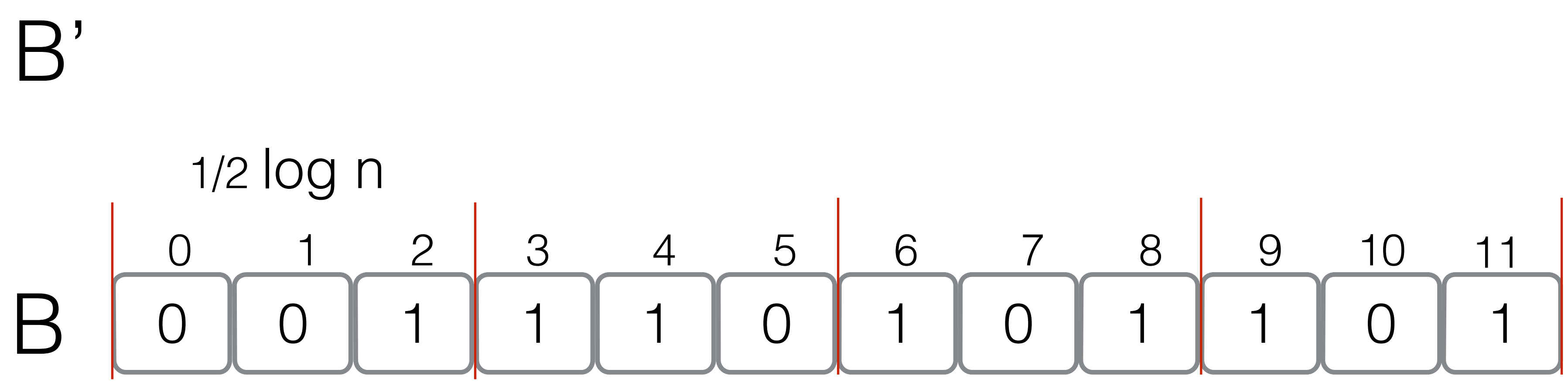
	0	1	2	3	4	5	6	7	8	9	10	11
B	0	0	1	1	1	0	1	0	1	1	0	1

Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_1(j)$ = # of 1 in B[0,j]

Space: n + O(n log log n/log n) bits
Query time: O(1)

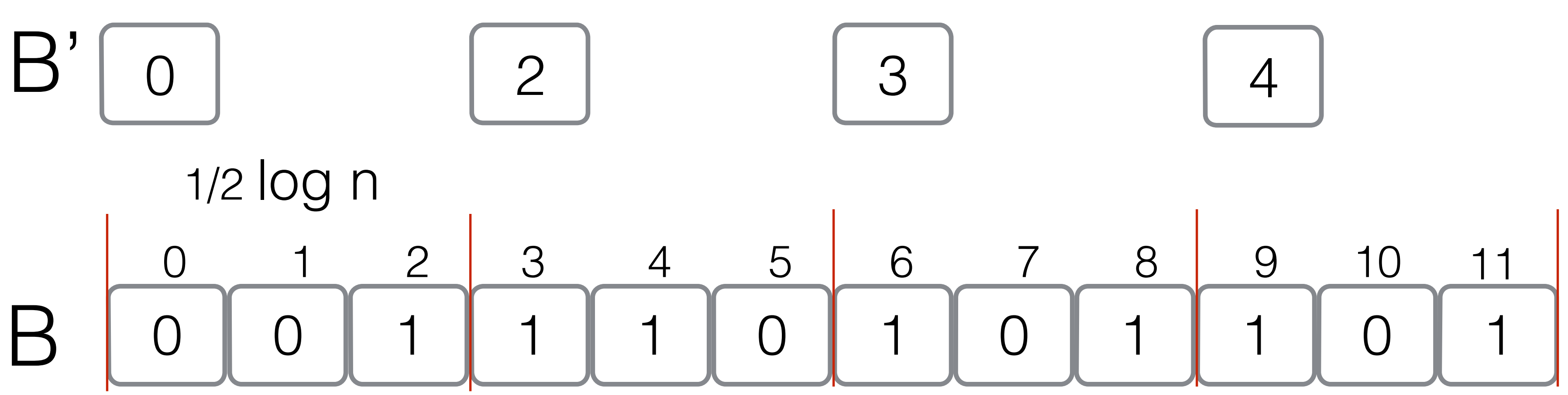


Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_1(j)$ = # of 1 in B[0,j]

Space: n + O(n log log n/log n) bits
Query time: O(1)



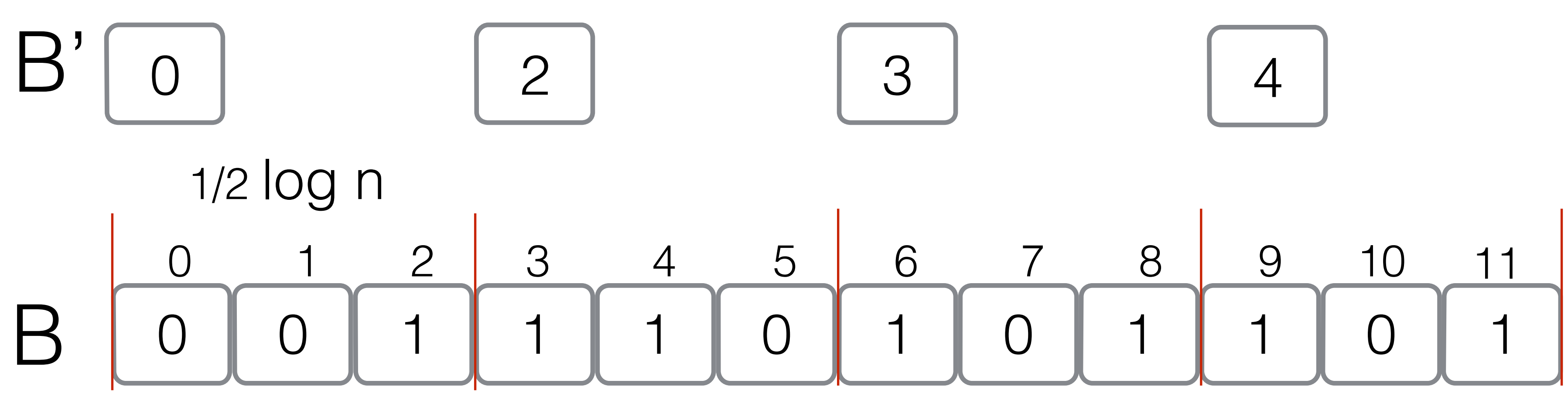
Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_1(j)$ = # of 1 in B[0,j]

Space: n + O(n log log n/log n) bits
Query time: O(1)

$Rank_0(7)$ =



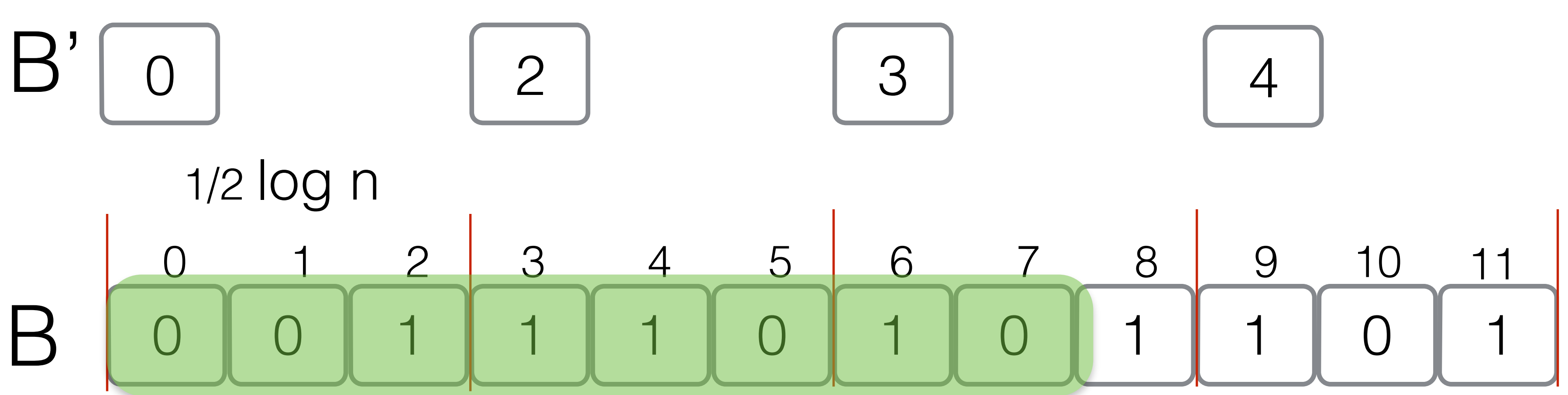
Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_1(j)$ = # of 1 in B[0,j]

Space: $n + O(n \log \log n / \log n)$ bits
Query time: $O(1)$

$Rank_0(7)$ =



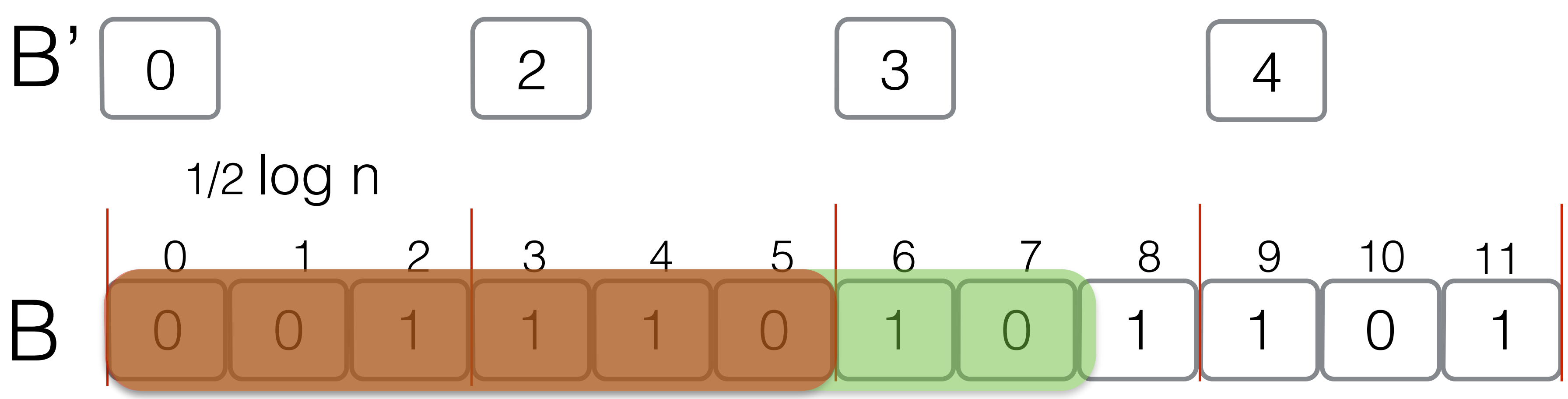
Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_1(j)$ = # of 1 in B[0,j]

Space: $n + O(n \log\log n / \log n)$ bits
Query time: O(1)

$Rank_0(7)$ =



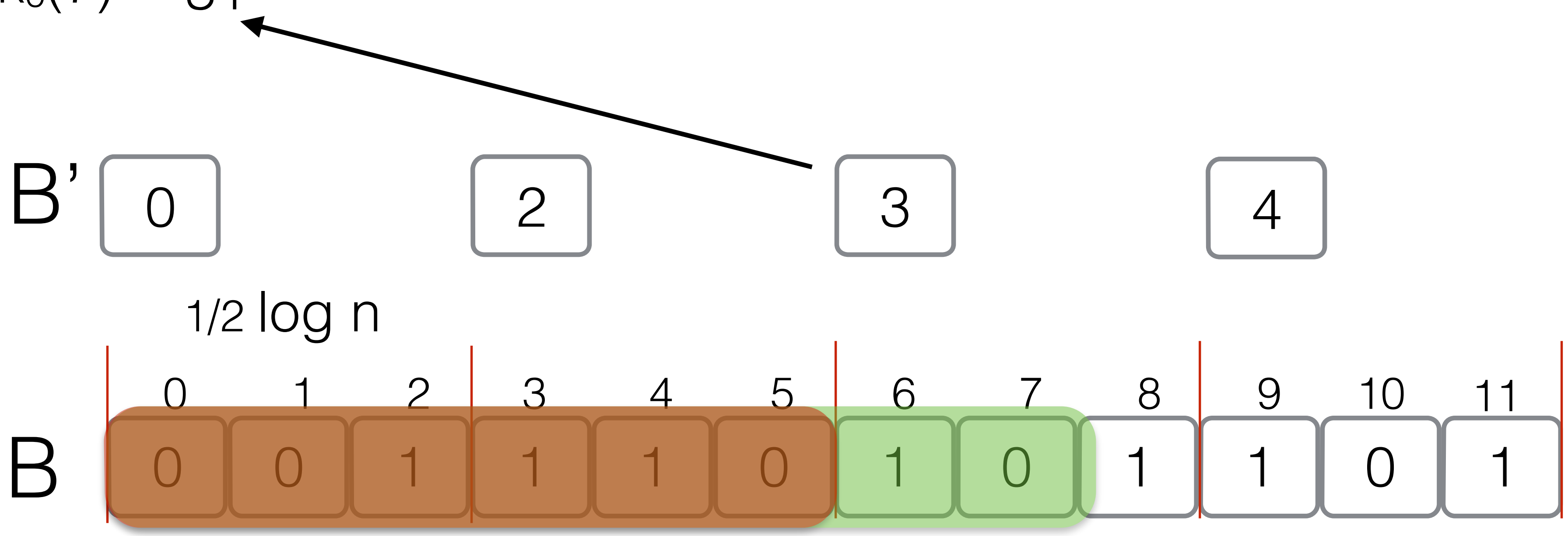
Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_1(j)$ = # of 1 in B[0,j]

Space: n + O(n log log n/log n) bits
Query time: O(1)

$Rank_0(7) = 3+$



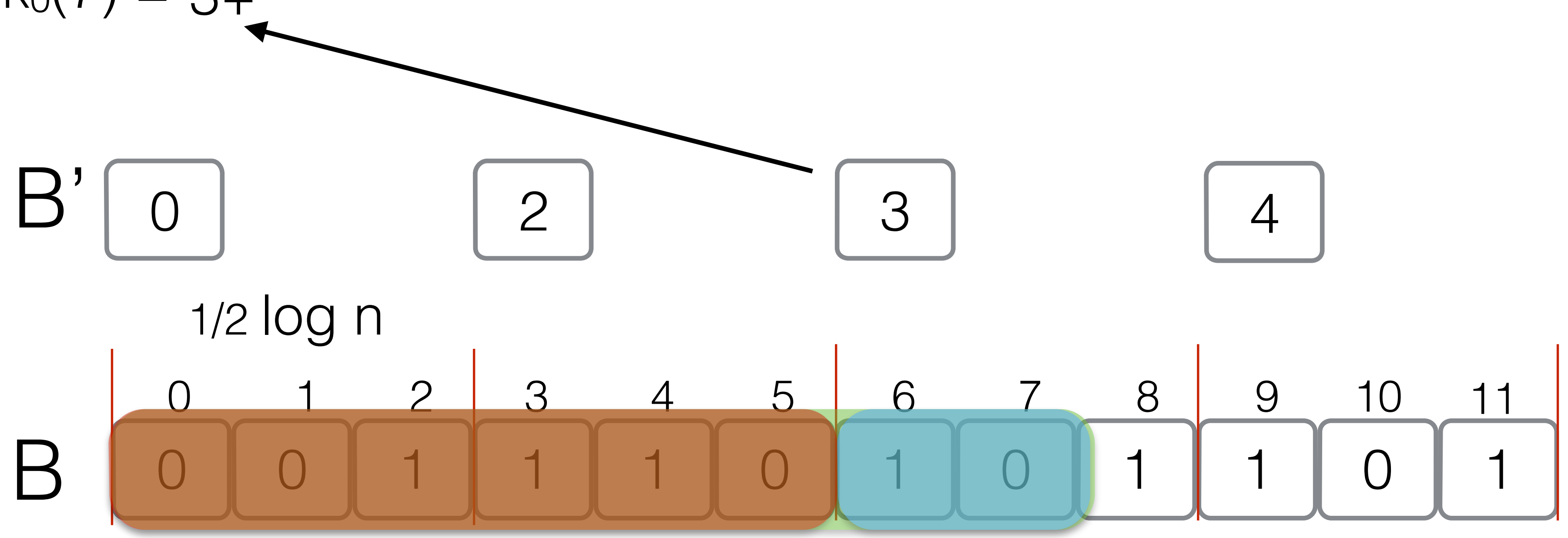
Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_1(j)$ = # of 1 in B[0,j]

Space: $n + O(n \log \log n / \log n)$ bits
Query time: $O(1)$

$Rank_0(7) = 3+$

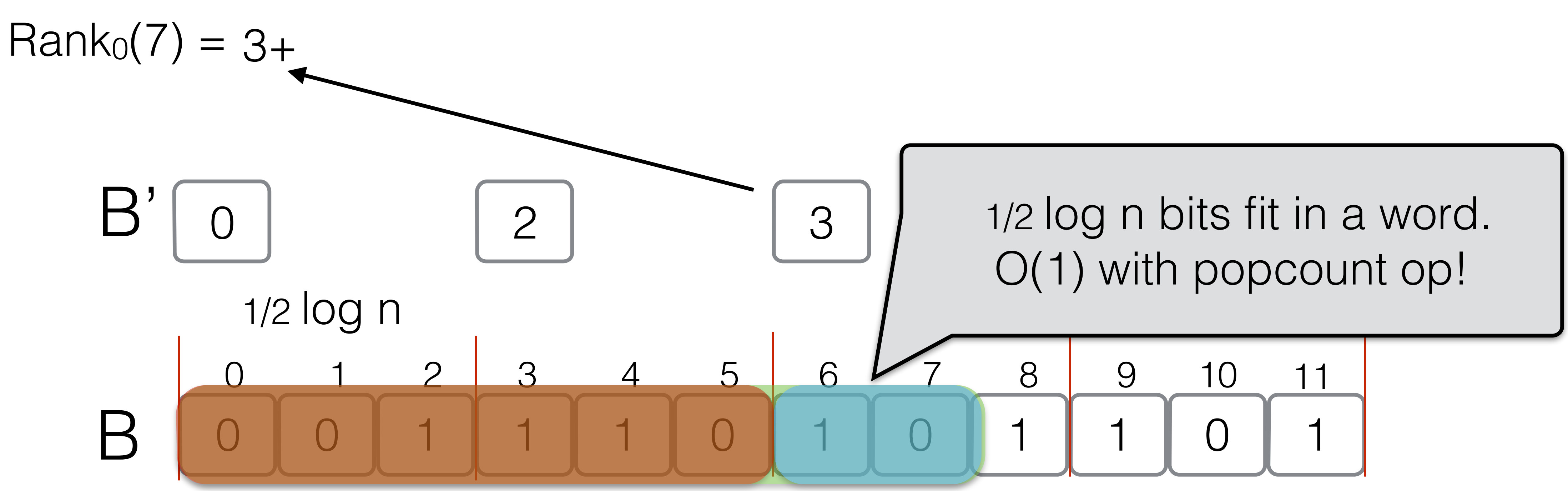


Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_1(j)$ = # of 1 in B[0,j]

Space: n + O(n log log n/log n) bits
Query time: O(1)



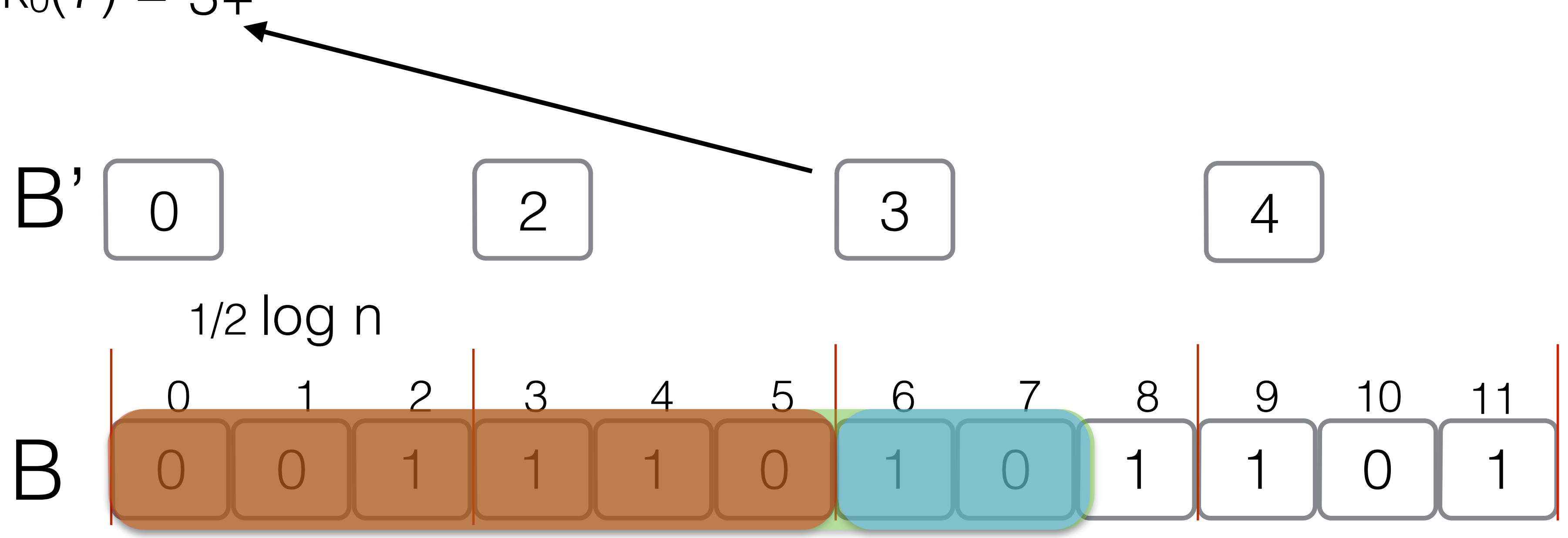
Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_1(j)$ = # of 1 in B[0,j]

Space: n + O(n log log n/log n) bits
Query time: O(1)

$Rank_0(7)$ = 3+



Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

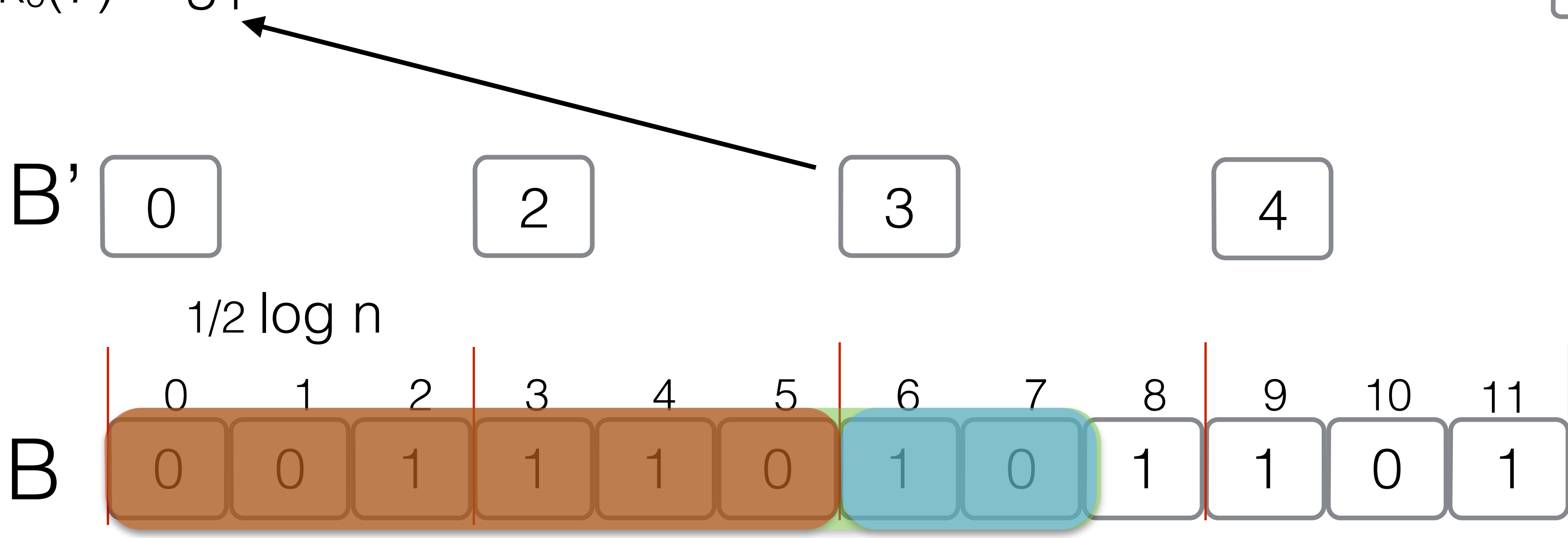
$Rank_1(j)$ = # of 1 in B[0,j]

> Space: n + O(n log log n/log n) bits
> Query time: O(1)

M

	1	2	3
000	1	2	3
001	1	2	2
010	1	1	2
011	1	1	1
100	0	1	2
101	0	1	1
110	0	0	1
111	0	0	0

$Rank_0(7)$ = 3+



Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

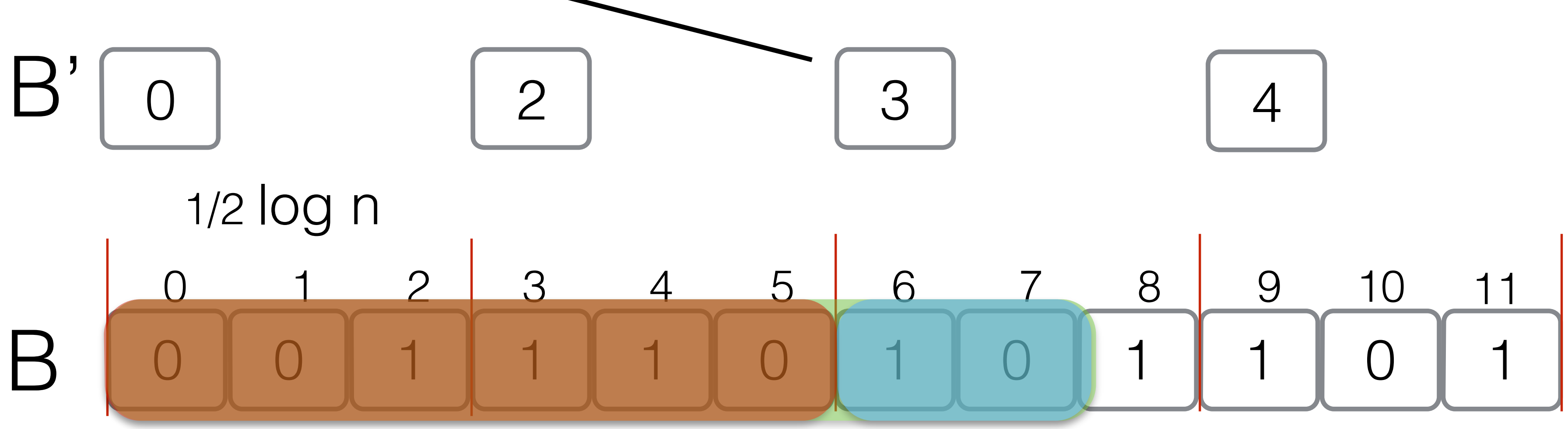
$Rank_1(j)$ = # of 1 in B[0,j]

> Space: $n + O(n \log \log n / \log n)$ bits
> Query time: $O(1)$

M

	1	2	3
000	1	2	3
001	1	2	2
010	1	1	2
011	1	1	1
100	0	1	2
101	0	1	1
110	0	0	1
111	0	0	0

$Rank_0(7) = 3+$



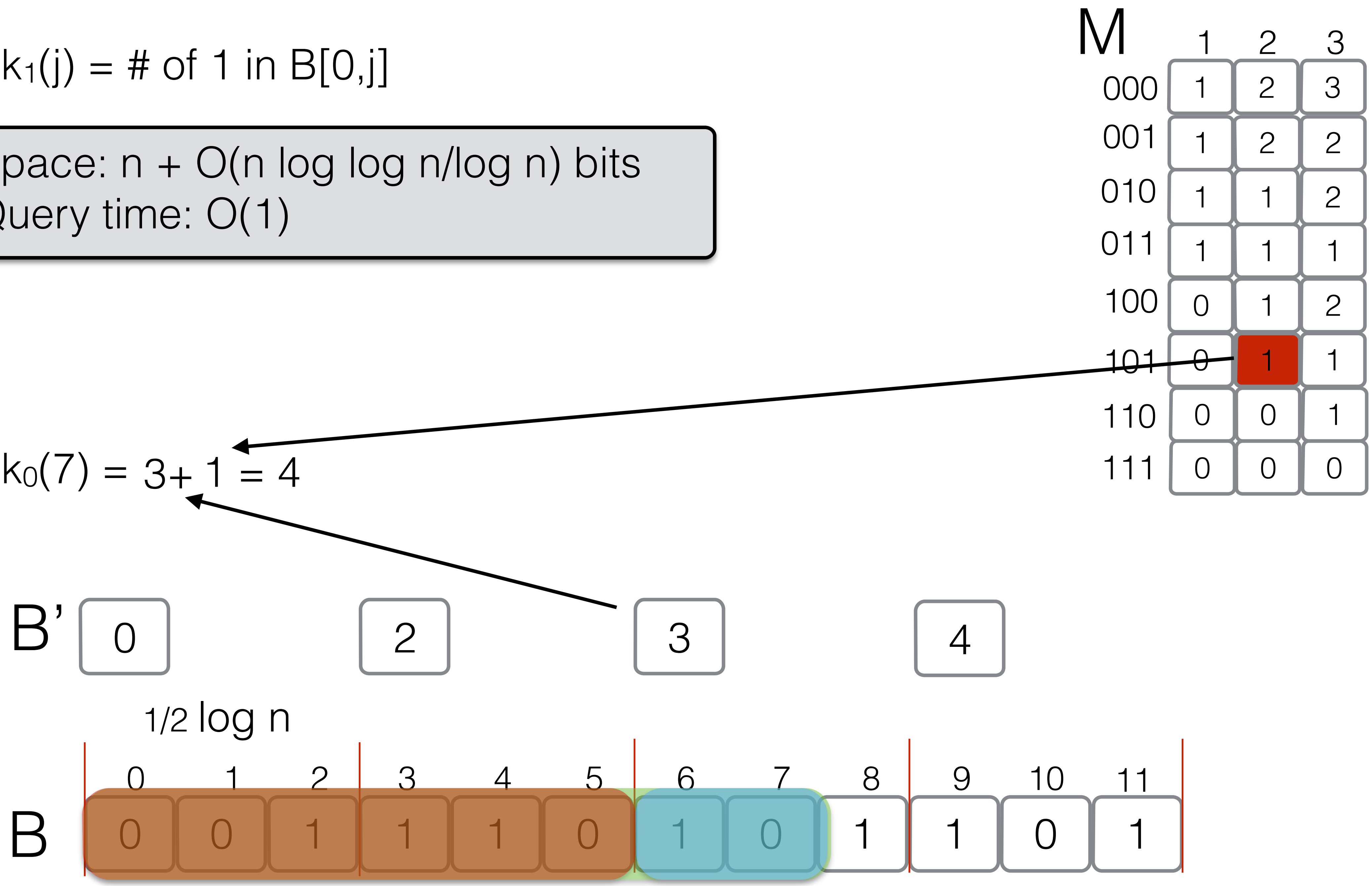
Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_1(j)$ = # of 1 in B[0,j]

> Space: $n + O(n \log \log n / \log n)$ bits
> Query time: $O(1)$

$Rank_0(7) = 3 + 1 = 4$



Rank/Select queries

How much space?

$Rank_0(j)$ = # of 0 in B[0,j]

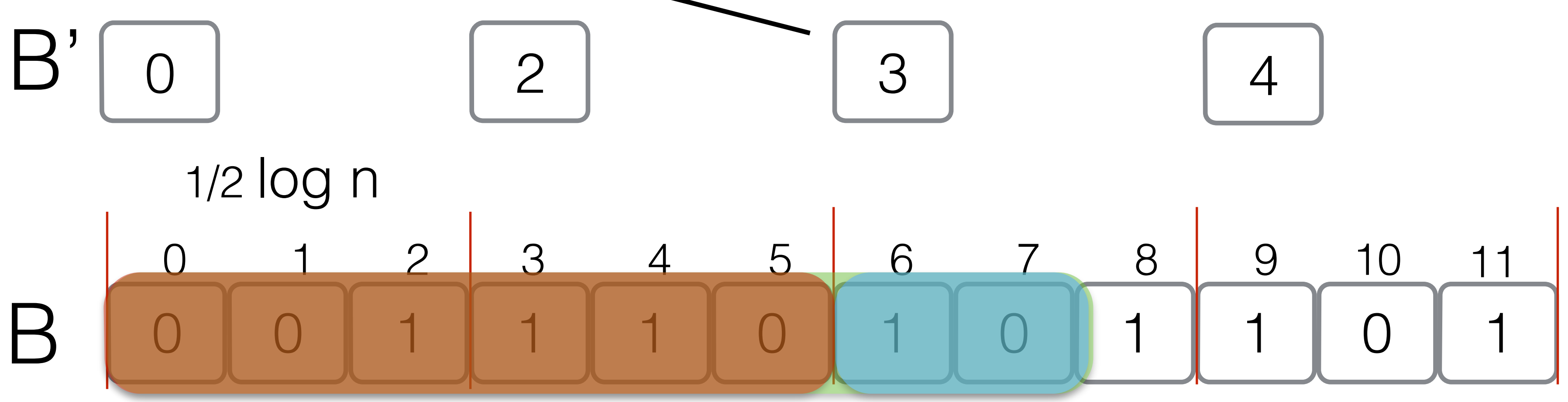
$Rank_1(j)$ = # of 1 in B[0,j]

Space: n + O(n log log n/log n) bits
Query time: O(1)

M

	1	2	3
000	1	2	3
001	1	2	2
010	1	1	2
011	1	1	1
100	0	1	2
101	0	1	1
110	0	0	1
111	0	0	0

$Rank_0(7) = 3 + 1 = 4$



Rank/Select queries

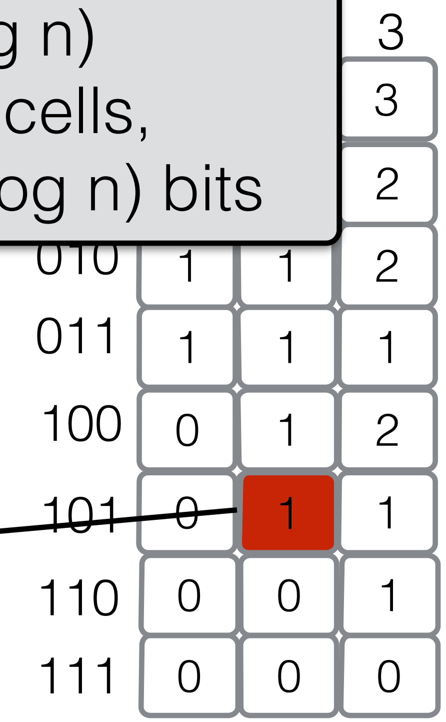
How much space?

$O(2^{1/2 \log n} \log n)$
$= O(\sqrt{n} \log n)$ cells,
each uses $O(\log \log n)$ bits

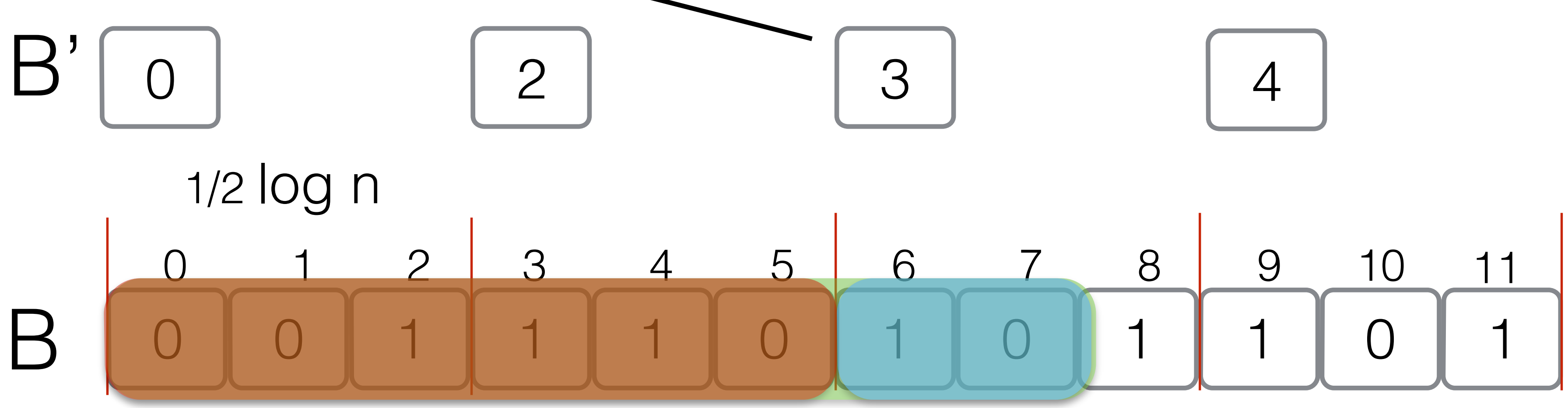
$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_1(j)$ = # of 1 in B[0,j]

Space: $n + O(n \log \log n / \log n)$ bits
Query time: $O(1)$



$Rank_0(7) = 3 + 1 = 4$



Rank/Select queries

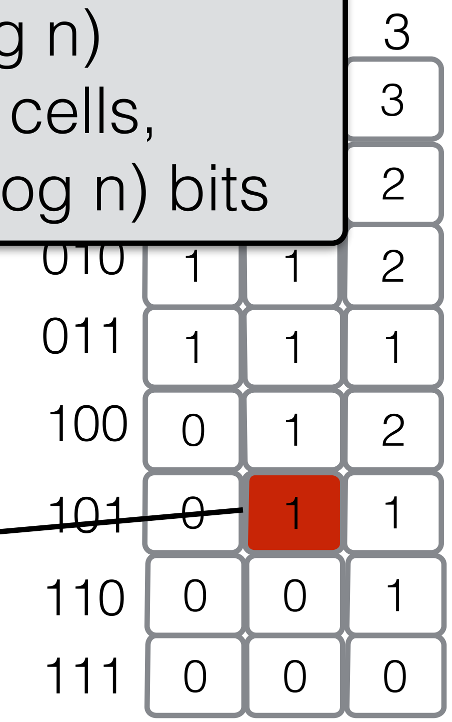
$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_1(j)$ = # of 1 in B[0,j]

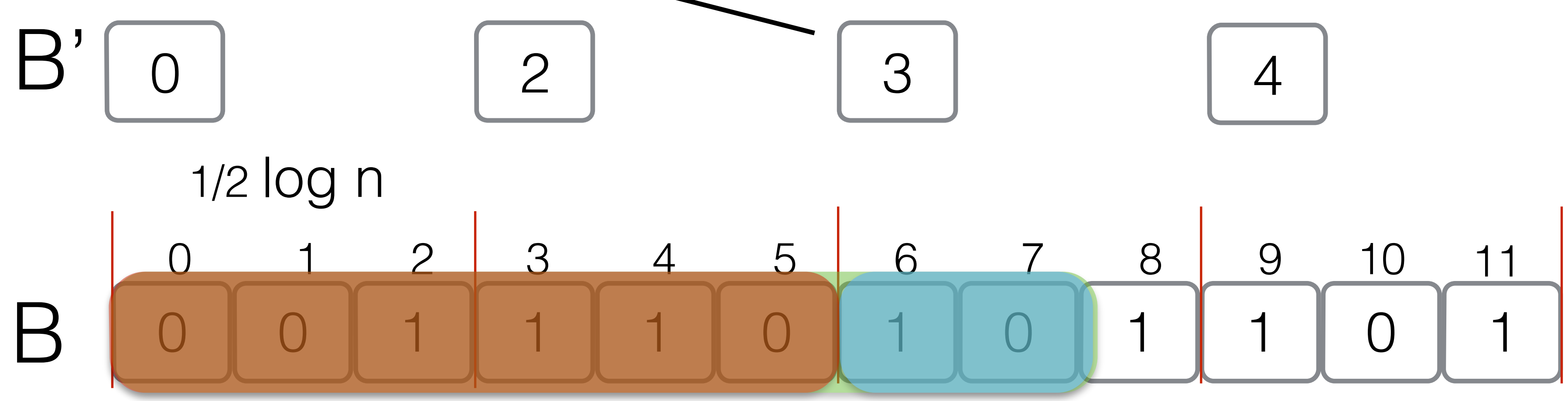
Space: n + O(n log log n/log n) bits
Query time: O(1)

How much space?

$O(2^{1/2 \log n} \log n)$
$= O(\sqrt{n} \log n)$ cells,
each uses O(log log n) bits



How much space?



Rank/Select queries

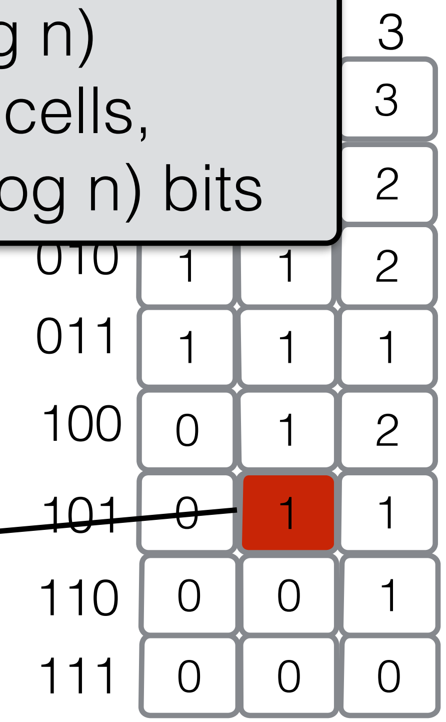
$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_1(j)$ = # of 1 in B[0,j]

Space: $n + O(n \log\log n / \log n)$ bits
Query time: $O(1)$

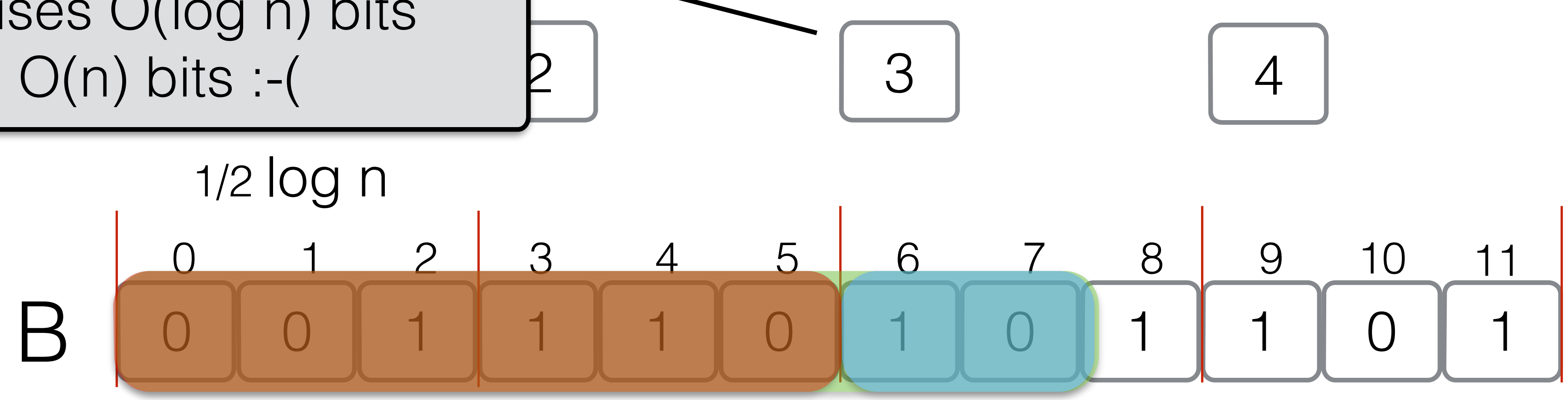
How much space?

$O(2^{1/2 \log n} \log n)$
$= O(\sqrt{n} \log n)$ cells,
each uses $O(\log\log n)$ bits



How much space?

$O(n/\log n)$ entries,
each uses $O(\log n)$ bits
$\Rightarrow O(n)$ bits :-(



Rank/Select queries

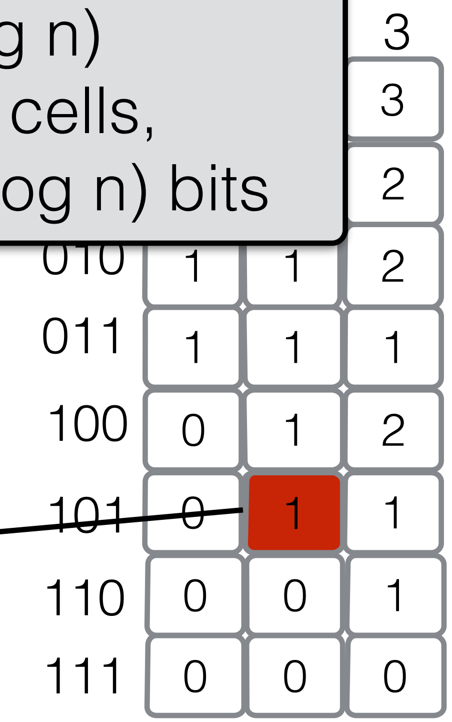
How much space?

$O(2^{1/2 \log n} \log n)$
$= O(\sqrt{n} \log n)$ cells,
each uses $O(\log \log n)$ bits

$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_1(j)$ = # of 1 in B[0,j]

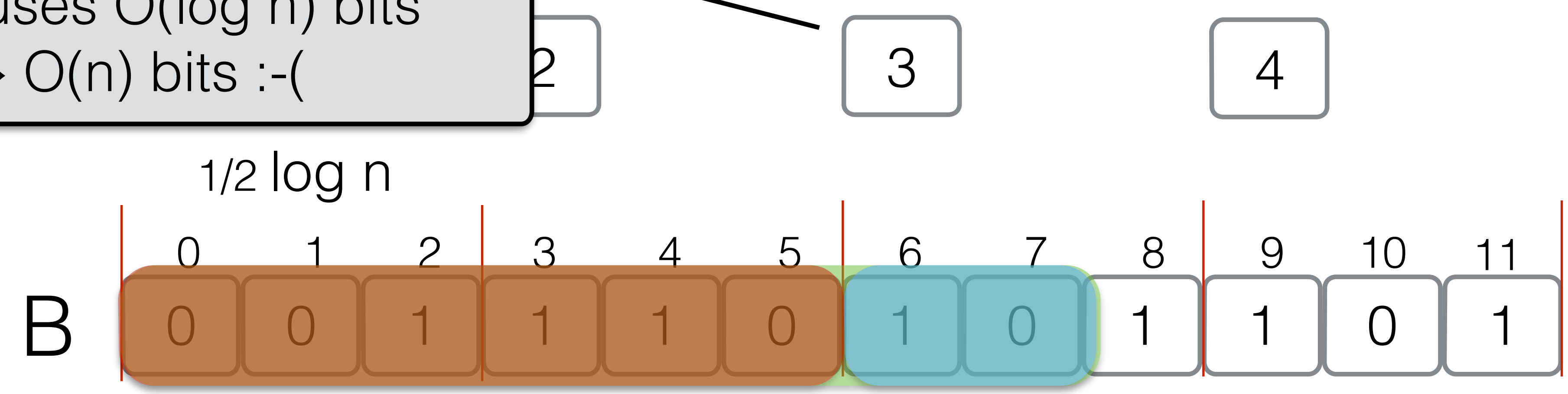
Space: $n + O(n \log \log n / \log n)$ bits
Query time: $O(1)$



How much space?

each uses $O(\log n)$ bits
$\Rightarrow O(n)$ bits :-(

Space: $O(n) + o(n)$ bits
Query time: $O(1)$



Rank/Select queries

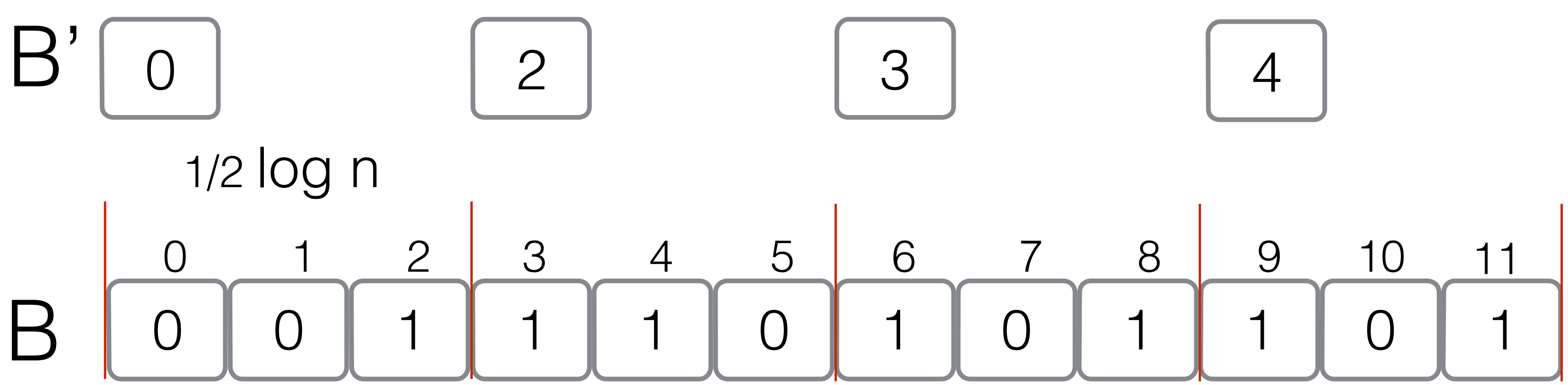
$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_1(j)$ = # of 1 in B[0,j]

Space: n + O(n log log n/log n) bits
Query time: O(1)

M

	1	2	3
000	1	2	3
001	1	2	2
010	1	1	2
011	1	1	1
100	0	1	2
101	0	1	1
110	0	0	1
111	0	0	0

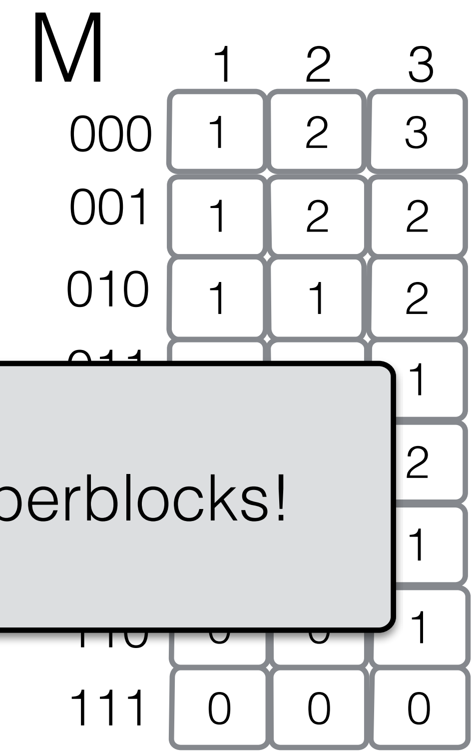


Rank/Select queries

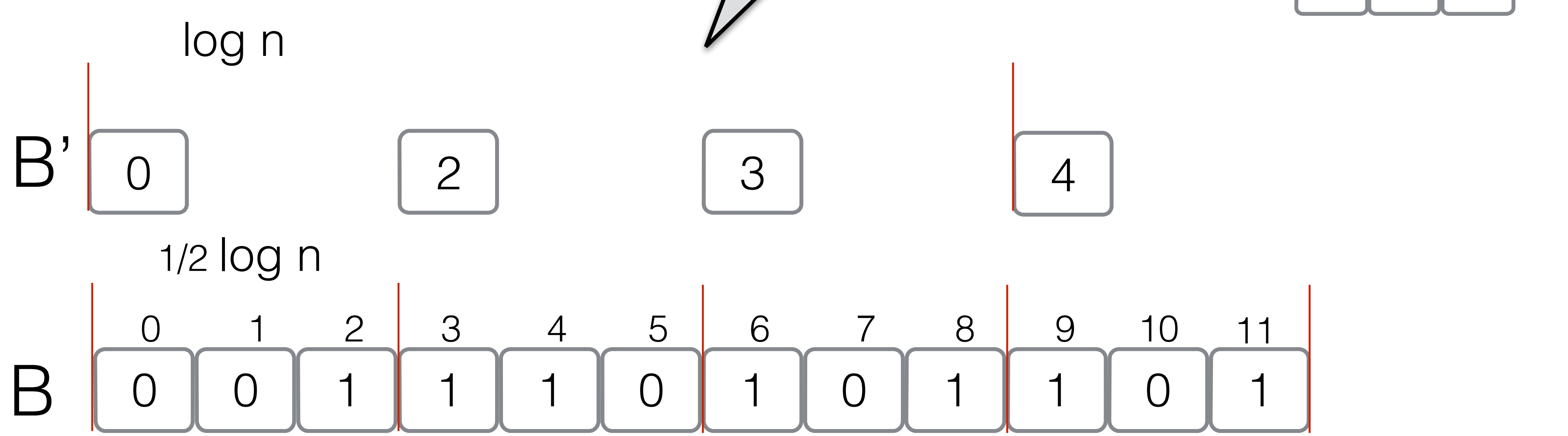
$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_1(j)$ = # of 1 in B[0,j]

Space: n + O(n log log n/log n) bits
Query time: O(1)



Groups into superblocks!



Rank/Select queries

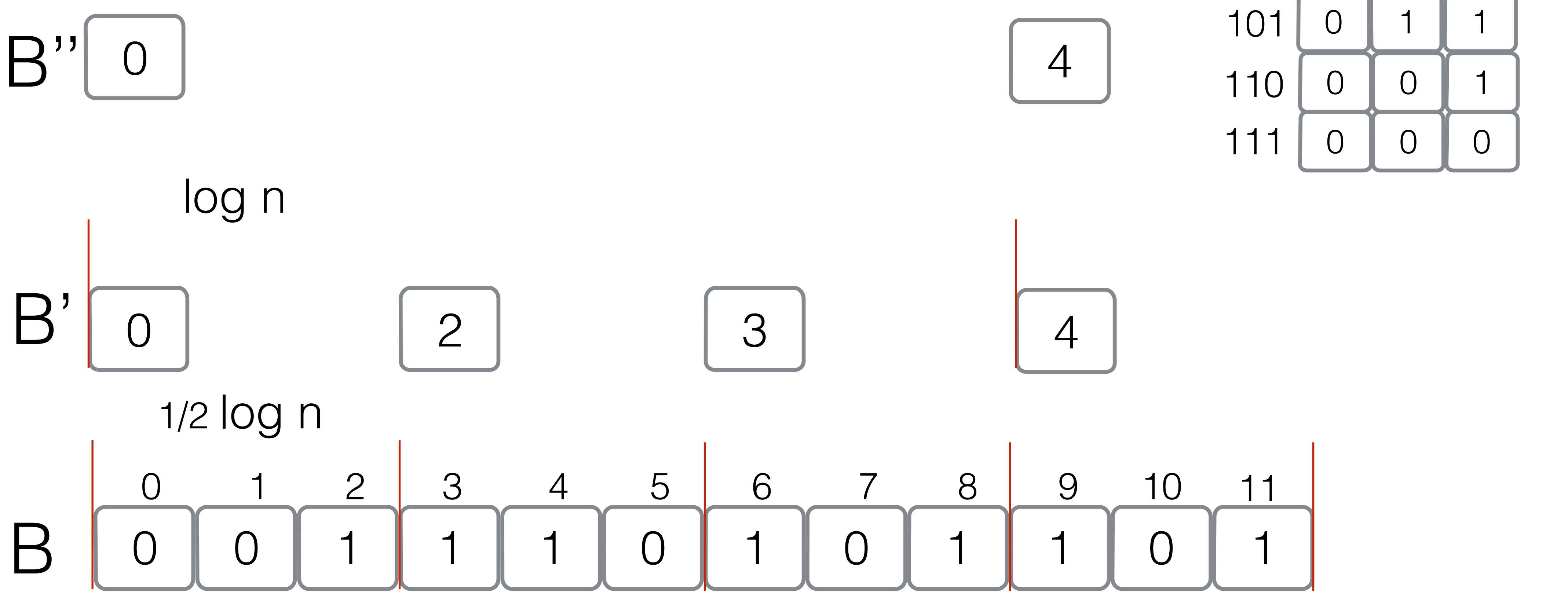
$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_1(j)$ = # of 1 in B[0,j]

Space: n + O(n log log n/log n) bits
Query time: O(1)

M

	1	2	3
000	1	2	3
001	1	2	2
010	1	1	2
011	1	1	1
100	0	1	2
101	0	1	1
110	0	0	1
111	0	0	0



Rank/Select queries

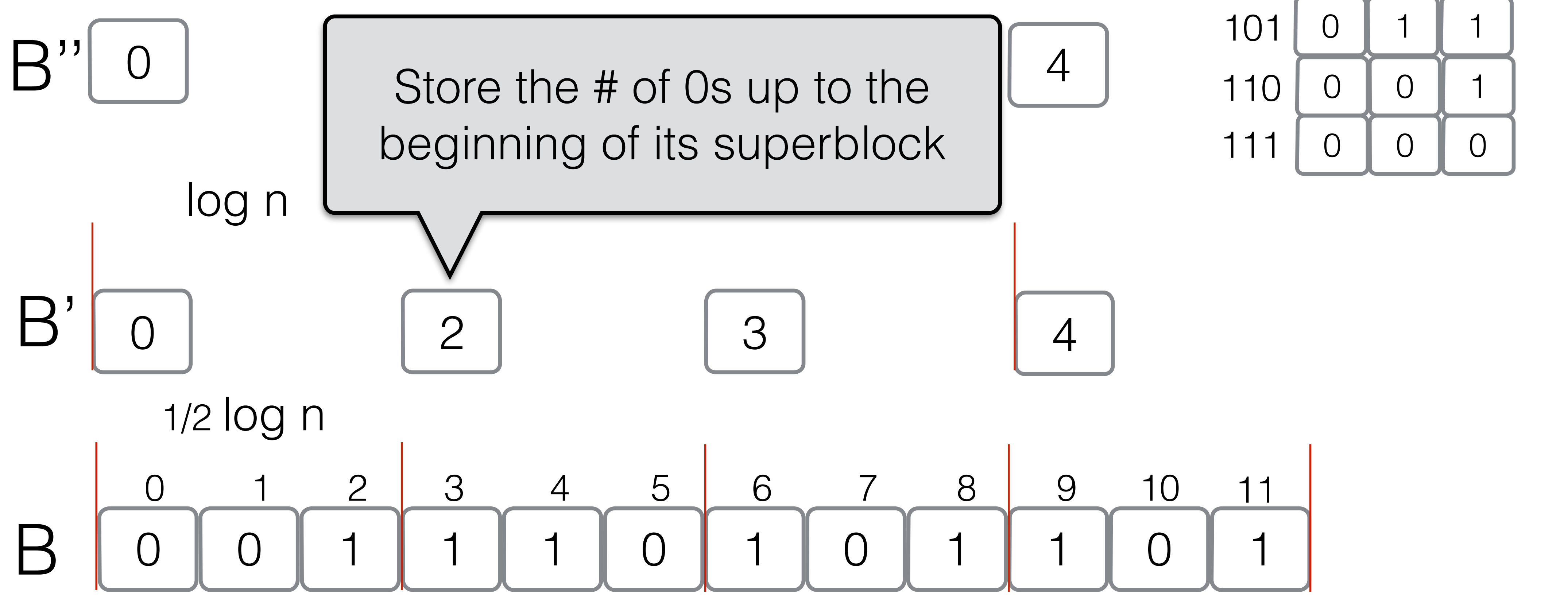
$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_1(j)$ = # of 1 in B[0,j]

Space: n + O(n log log n/log n) bits
Query time: O(1)

M

	1	2	3
000	1	2	3
001	1	2	2
010	1	1	2
011	1	1	1
100	0	1	2
101	0	1	1
110	0	0	1
111	0	0	0



Rank/Select queries

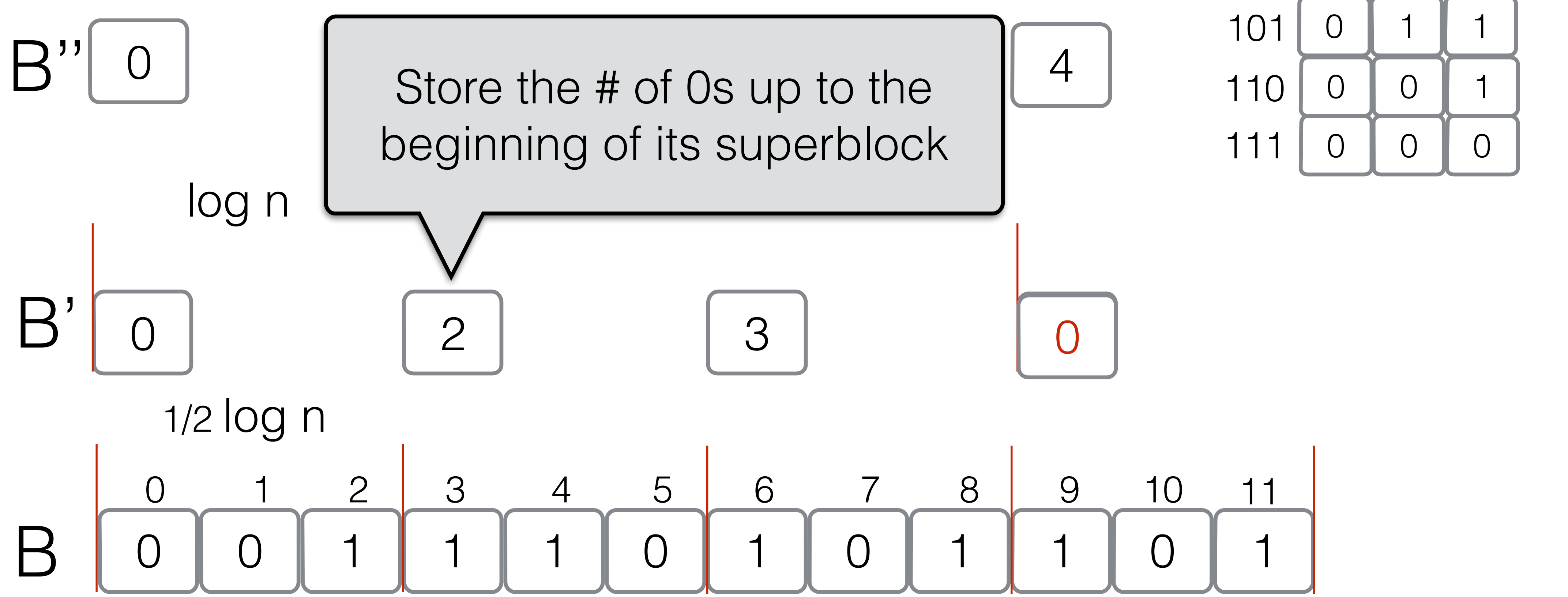
$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_1(j)$ = # of 1 in B[0,j]

Space: n + O(n log log n/log n) bits
Query time: O(1)

M

	1	2	3
000	1	2	3
001	1	2	2
010	1	1	2
011	1	1	1
100	0	1	2
101	0	1	1
110	0	0	1
111	0	0	0



Rank/Select queries

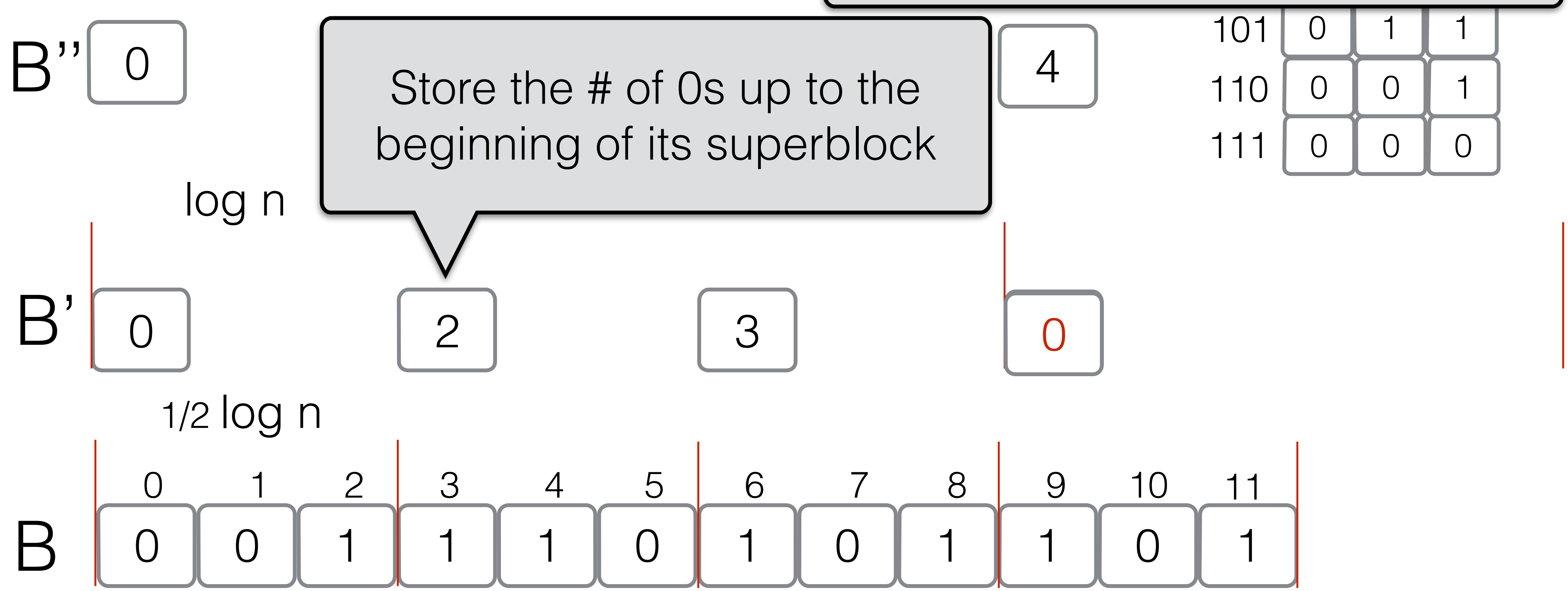
$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_1(j)$ = # of 1 in B[0,j]

Space: n + O(n log log n/log n) bits
Query time: O(1)

$Rank_0(j)$ is split into 3 parts:

- # of 0s up to the superblock of j
- # of 0s up to the block of j
- # of 0s within the block of j



Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

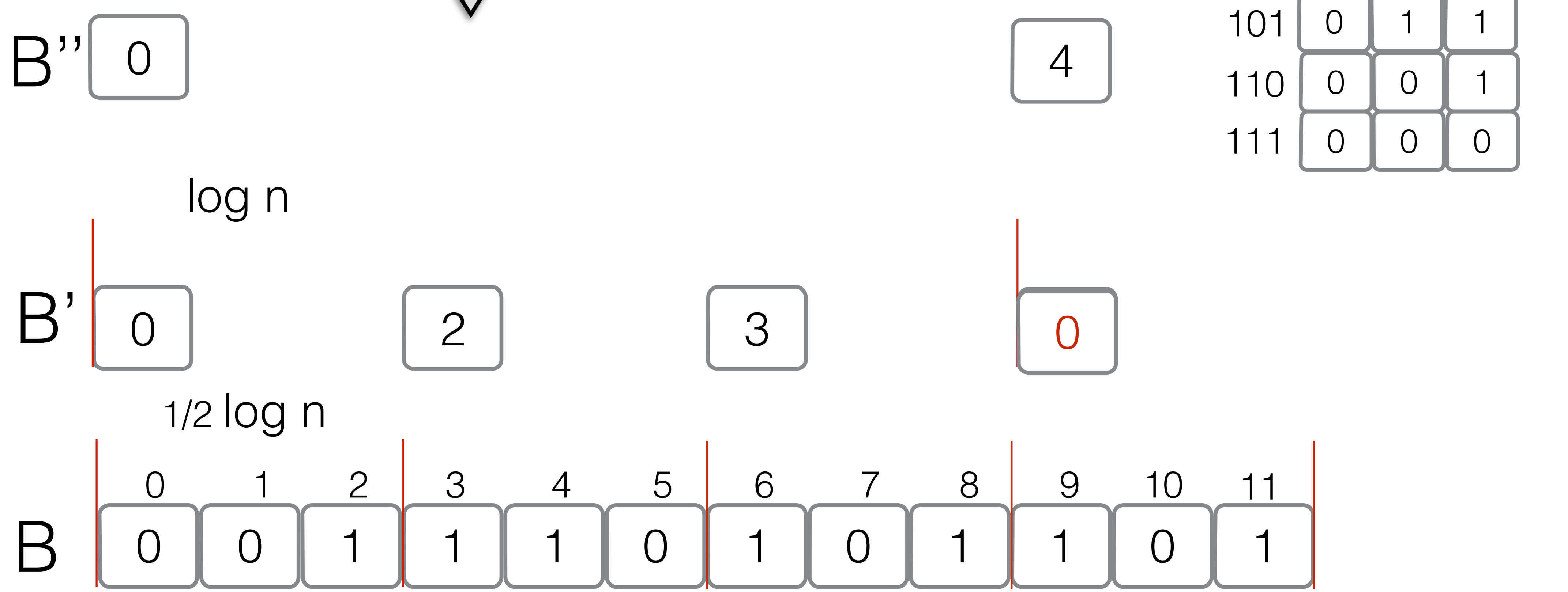
$Rank_1(j)$ = # of 1 in B[0,j]

) bits

How much space?

M

	1	2	3
000	1	2	3
001	1	2	2
010	1	1	2
011	1	1	1
100	0	1	2
101	0	1	1
110	0	0	1
111	0	0	0



Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_1(j)$ = # of 1 in B[0,j]

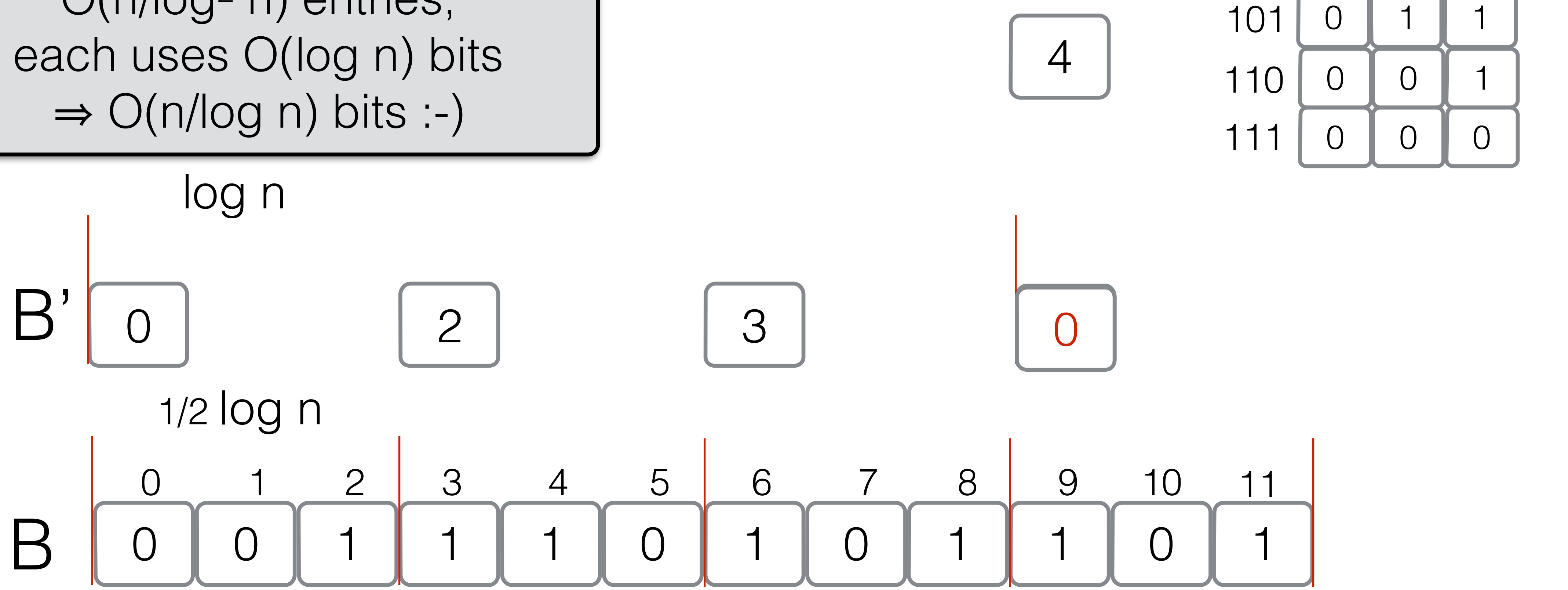
) bits

How much space?

O(n/log² n) entries,
each uses O(log n) bits
⇒ O(n/log n) bits :-)

M

	1	2	3
000	1	2	3
001	1	2	2
010	1	1	2
011	1	1	1
100	0	1	2
101	0	1	1
110	0	0	1
111	0	0	0



Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

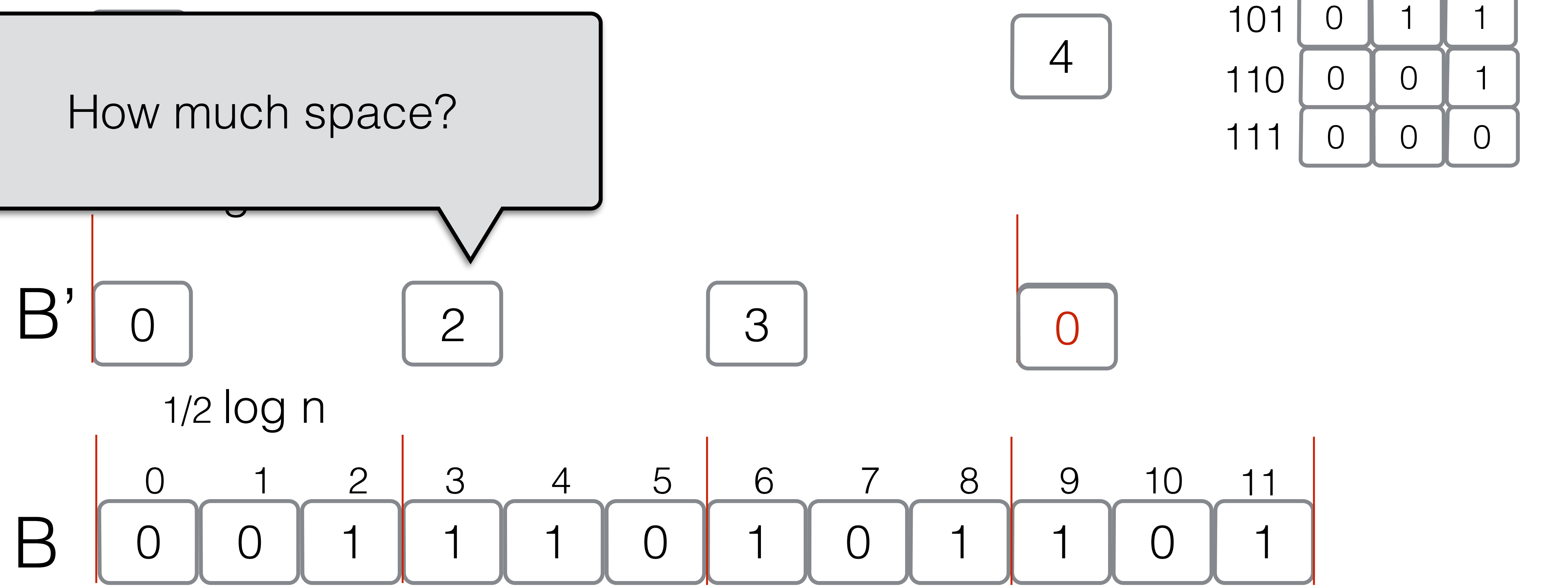
$Rank_1(j)$ = # of 1 in B[0,j]

Space: n + O(n log log n/log n) bits
Query time: O(1)

How much space?

M

	1	2	3
000	1	2	3
001	1	2	2
010	1	1	2
011	1	1	1
100	0	1	2
101	0	1	1
110	0	0	1
111	0	0	0



Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_1(j)$ = # of 1 in B[0,j]

> Space: n + O(n log log n/log n) bits
> Query time: O(1)

> How much space?

> O(n/log n) entries,
> each uses O(log log n) bits
> ⇒ O(n log log n/log n) bits :-)

M

	1	2	3
000	1	2	3
001	1	2	2
010	1	1	2
011	1	1	1
100	0	1	2
101	0	1	1
110	0	0	1
111	0	0	0

4

3

0

B

	0	1	2	3	4	5	6	7	8	9	10	11
B	0	0	1	1	1	0	1	0	1	1	0	1

Rank/Select queries

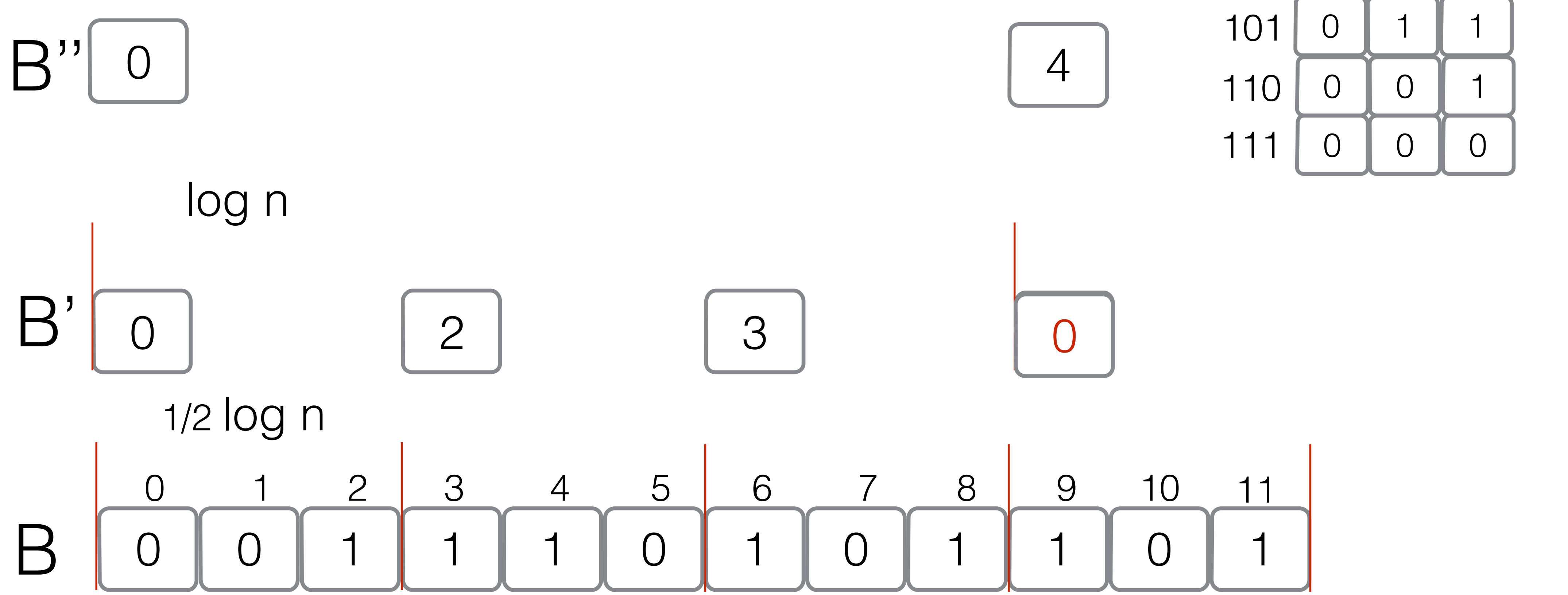
$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_1(j)$ = # of 1 in B[0,j]

Space: n + O(n log log n/log n) bits
Query time: O(1)

M

	1	2	3
000	1	2	3
001	1	2	2
010	1	1	2
011	1	1	1
100	0	1	2
101	0	1	1
110	0	0	1
111	0	0	0



Rank/Select queries

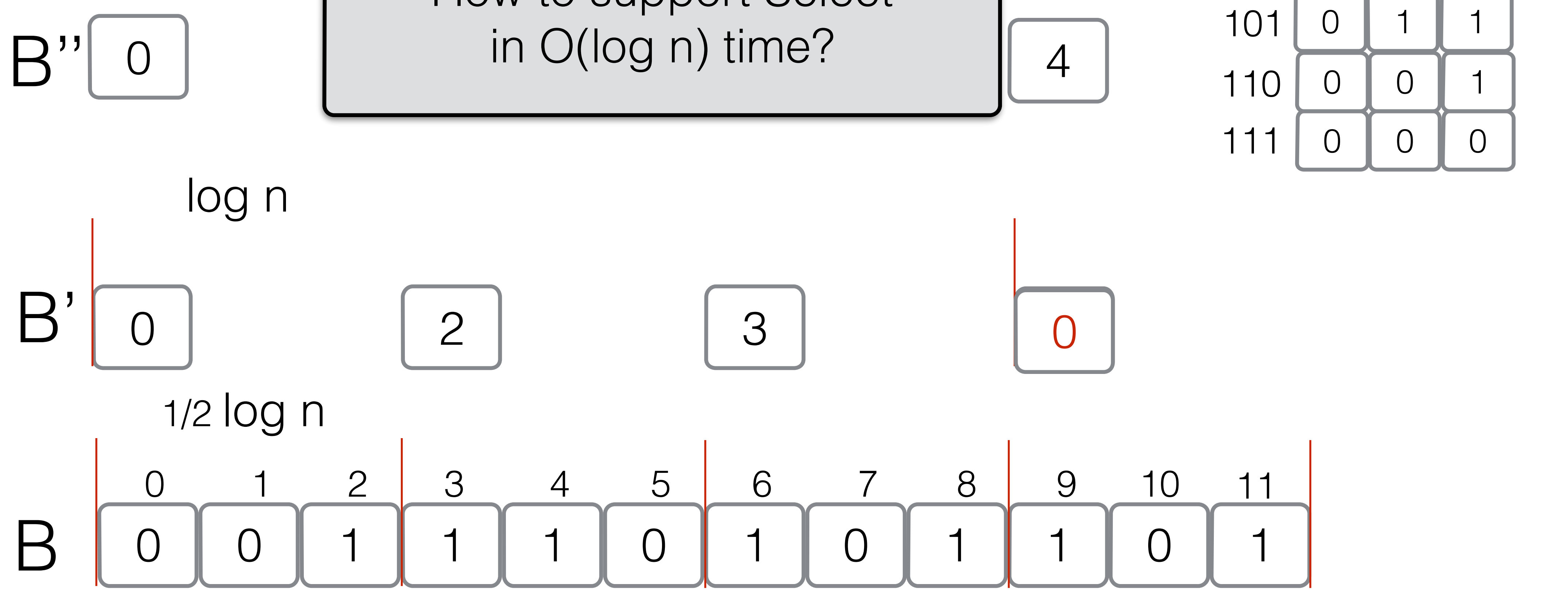
$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_1(j)$ = # of 1 in B[0,j]

Space: n + O(n log log n/log n) bits
Query time: O(1)

How to support Select in O(log n) time?

M	1	2	3
000	1	2	3
001	1	2	2
010	1	1	2
011	1	1	1
100	0	1	2
101	0	1	1
110	0	0	1
111	0	0	0



Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_1(j)$ = # of 1 in B[0,j]

Space: $n + O(n \log\log n / \log n)$ bits
Query time: $O(1)$

How to support Select in $O(\log n)$ time?

Select can be solved in $O(1)$ time with a more difficult approach

M	1	2	3
000	1	2	3
001	1	2	2
010	1	1	2
011	1	1	1
100	0	1	2
101	0	1	1
110	0	0	1
111	0	0	0

B'': 0 … 4 (log n)

B': 0 … 0 (1/2 log n)

	0	1	2	3	4	5	6	7	8	9	10	11
B	0	0	1	1	1	0	1	0	1	1	0	1

Elias-Fano representation

Given a sequence of n (positive) integers summing up to m

Elias-Fano representation

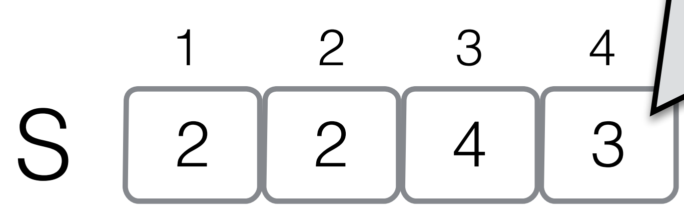
Given a sequence of n (positive) integers summing up to m

	1	2	3	4
S	2	2	4	3

Elias-Fano representation

Given a sequence of n (positive) integers summing up to m

Trivial representation requires
O(n log m) bits :-(
Can we do better?



Elias-Fano representation

Given a sequence of n (positive) integers summing up to m

	1	2	3	4
S	2	2	4	3

Represent the integer x by writing x-1 in unary to obtain B of m bits with n zeros

B

Elias-Fano representation

Given a sequence of n (positive) integers summing up to m

S

1	2	3	4
2	2	4	3

Represent the integer x by writing x-1 in unary to obtain B of m bits with n zeros

B

0	1
1	0

Elias-Fano representation

Given a sequence of n (positive) integers summing up to m

	1	2	3	4
S	2	2	4	3

Represent the integer x by writing x-1 in unary to obtain B of m bits with n zeros

	0	1	2	3
B	1	0	1	0

Elias-Fano representation

Given a sequence of n (positive) integers summing up to m

	1	2	3	4
S	2	2	4	3

Represent the integer x by writing x-1 in unary to obtain B of m bits with n zeros

	0	1	2	3	4	5	6	7
B	1	0	1	0	1	1	1	0

Elias-Fano representation

Given a sequence of n (positive) integers summing up to m

S

1	2	3	4
2	2	4	3

Represent the integer x by writing x-1 in unary to obtain B of m bits with n zeros

B

0	1	2	3	4	5	6	7	8	9	10
1	0	1	0	1	1	1	0	1	1	0

Elias-Fano representation

Given a sequence of n (positive) integers summing up to m

S

1	2	3	4
2	2	4	3

The i-th value of S is $Select_0(i)$-$Select_0(i-1)$

Represent the integer x by writing x-1 in unary to obtain B of m bits with n zeros

B

0	1	2	3	4	5	6	7	8	9	10
1	0	1	0	1	1	1	0	1	1	0

Elias-Fano representation

Given a sequence of n (positive) integers summing up to m

S

1	2	3	4
2	2	4	3

The i-th value of S is $Select_0(i)$-$Select_0(i-1)$

Represent the integer x by writing x-1 in unary to obtain B of m bits with n zeros

B

0	1	2	3	4	5	6	7	8	9	10
1	0	1	0	1	1	1	0	1	1	0

Elias-Fano representation

Given a sequence of n (positive) integers summing up to m

	1	2	3	4
S	2	2	4	3

The i-th value of S is $Select_0(i)-Select_0(i-1)$

Represent the integer x by writing x-1 in unary to obtain B of m bits with n zeros

$Select_0(2)=3$ $Select_0(3)=7$

	0	1	2	3	4	5	6	7	8	9	10
B	1	0	1	0	1	1	1	0	1	1	0

Elias-Fano representation

Given a sequence of n (positive) integers summing up to m

Space: n log (m/n) + O(n) bits
$Select_0$ in O(1)

See Lecture 6 (10/10/2013)

	1	2	3	4
S	2	2	4	3

The i-th value of S is $Select_0(i)-Select_0(i-1)$

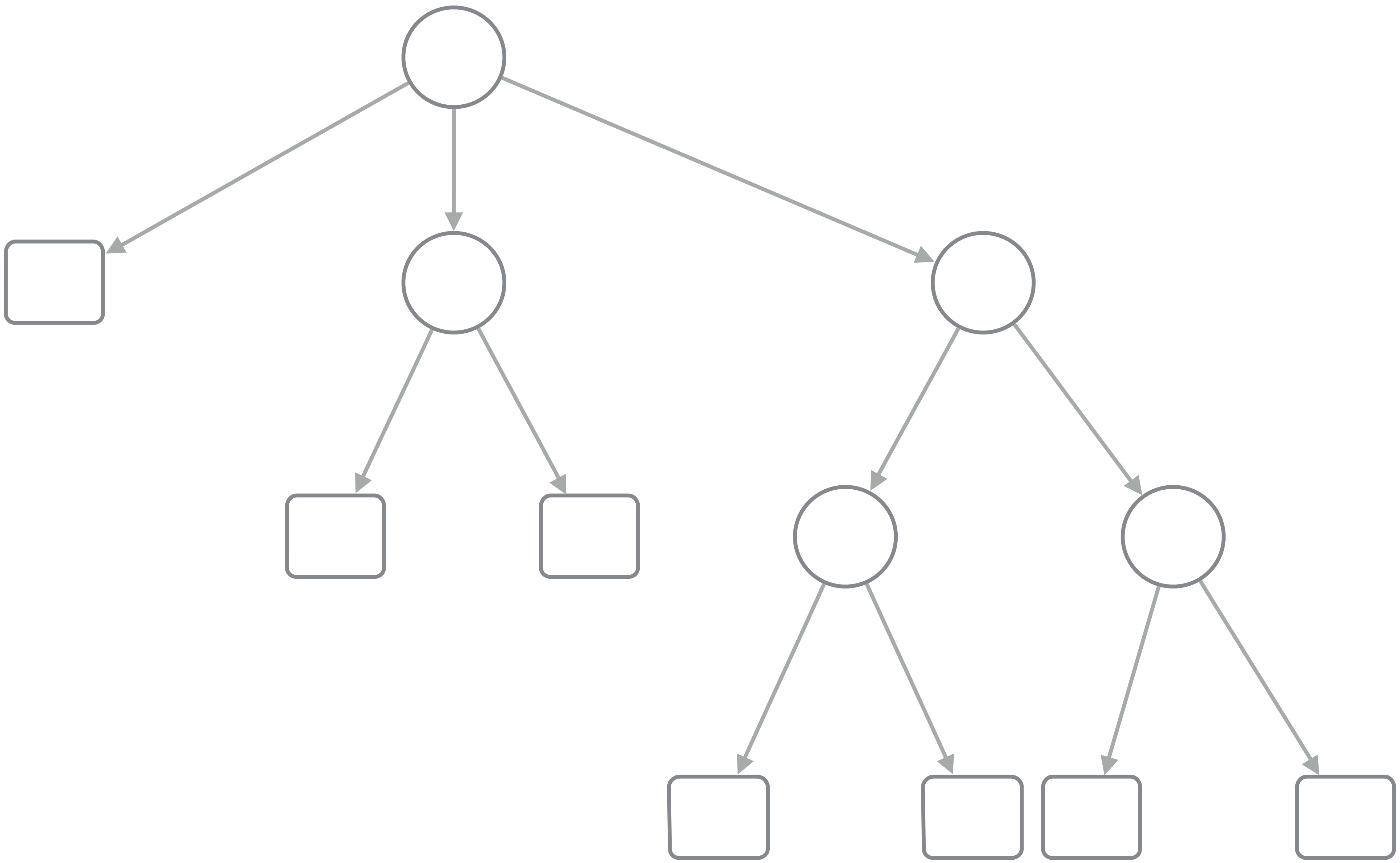
Represent the integer x by writing x-1 in unary to obtain B of m bits with n zeros

$Select_0(2)=3$ $Select_0(3)=7$

	0	1	2	3	4	5	6	7	8	9	10
B	1	0	1	0	1	1	1	0	1	1	0

Succinct representation of trees (1)

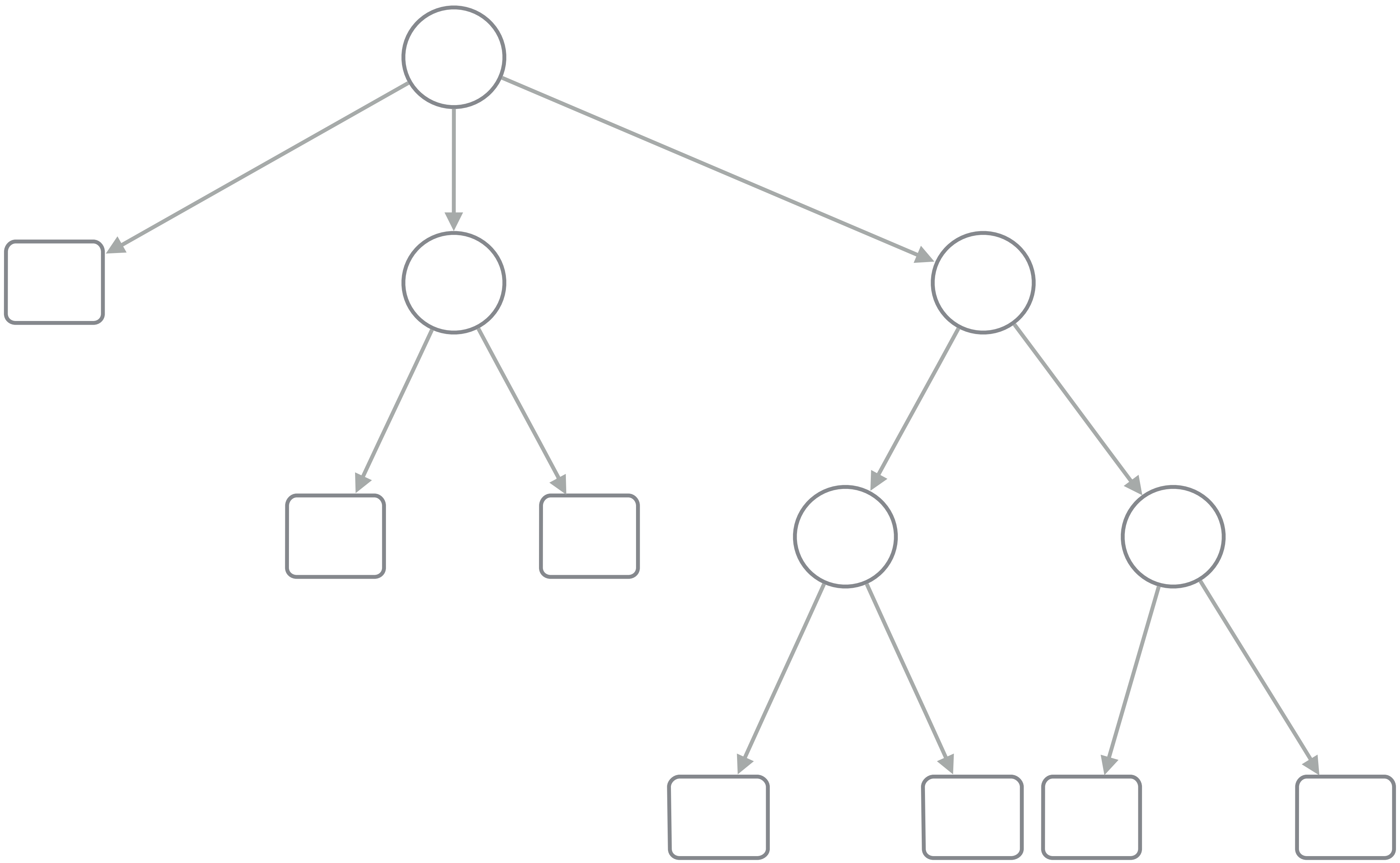
[LOUDS - Level-order unary degree sequence]



Succinct representation of trees (1)

[LOUDS - Level-order unary degree sequence]

Trivial: O(n log n) bits

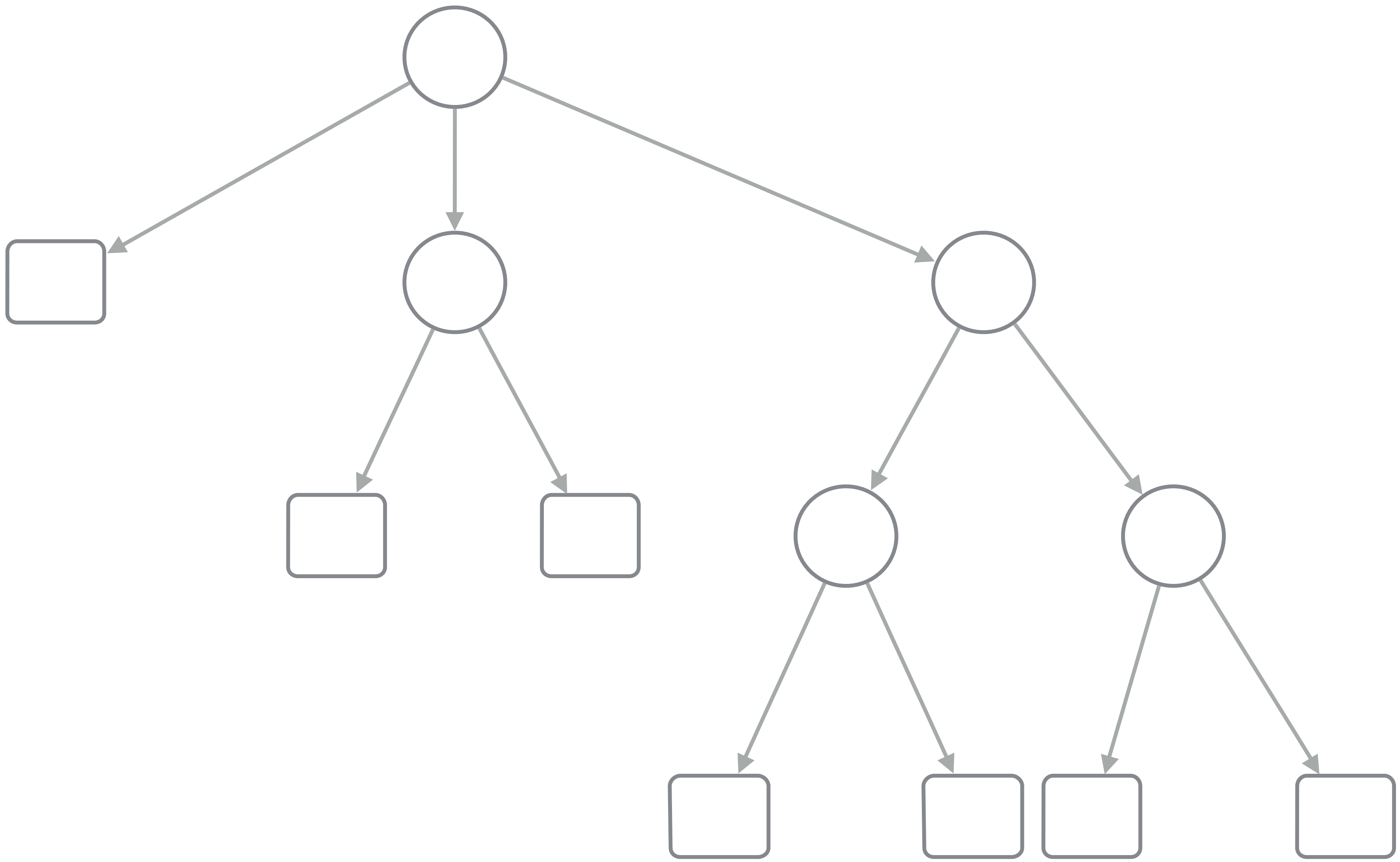


Succinct representation of trees (1)

[LOUDS - Level-order unary degree sequence]

Trivial: O(n log n) bits

Best: 2n bits

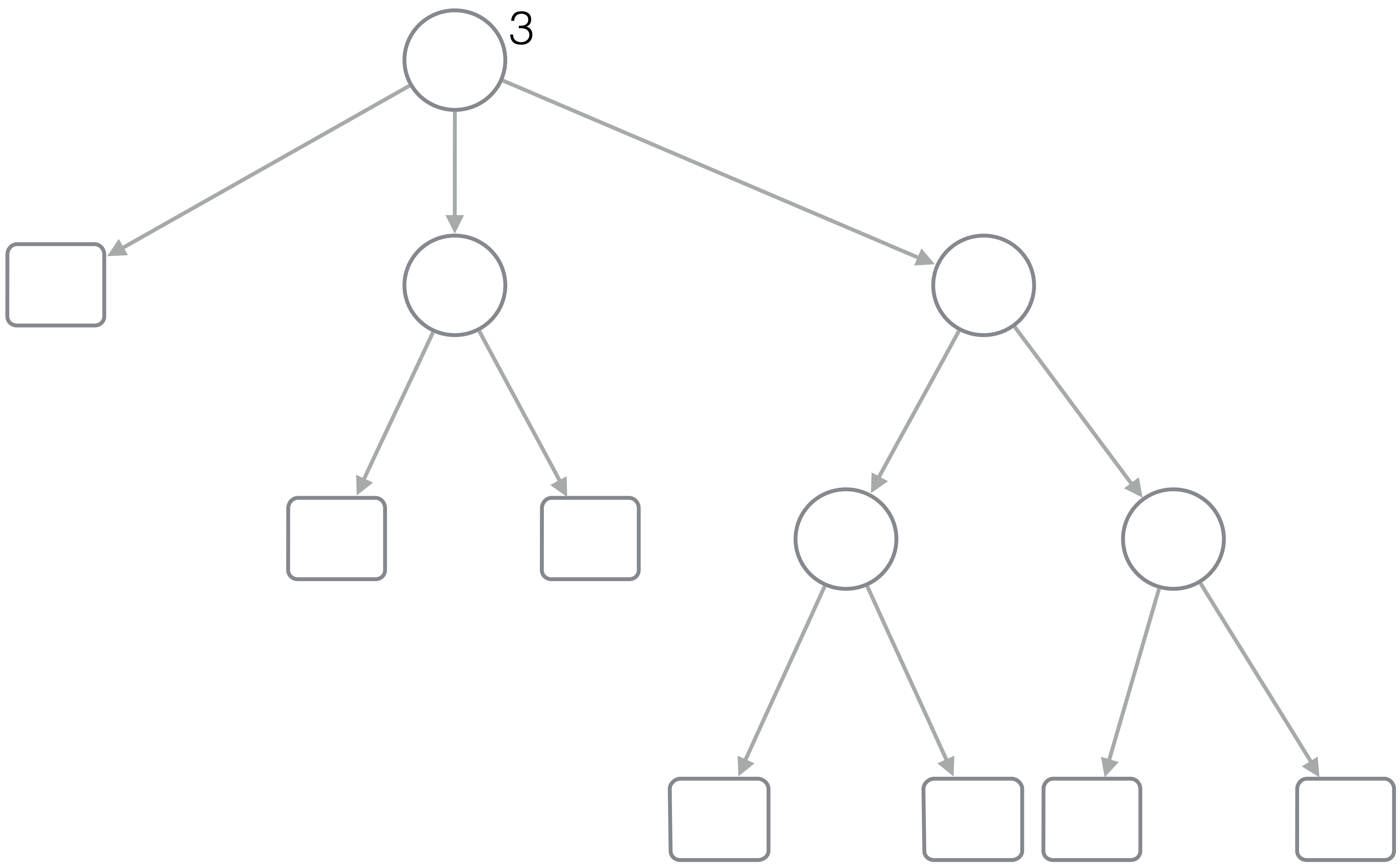


Succinct representation of trees (1)

[LOUDS - Level-order unary degree sequence]

Trivial: O(n log n) bits

Best: 2n bits

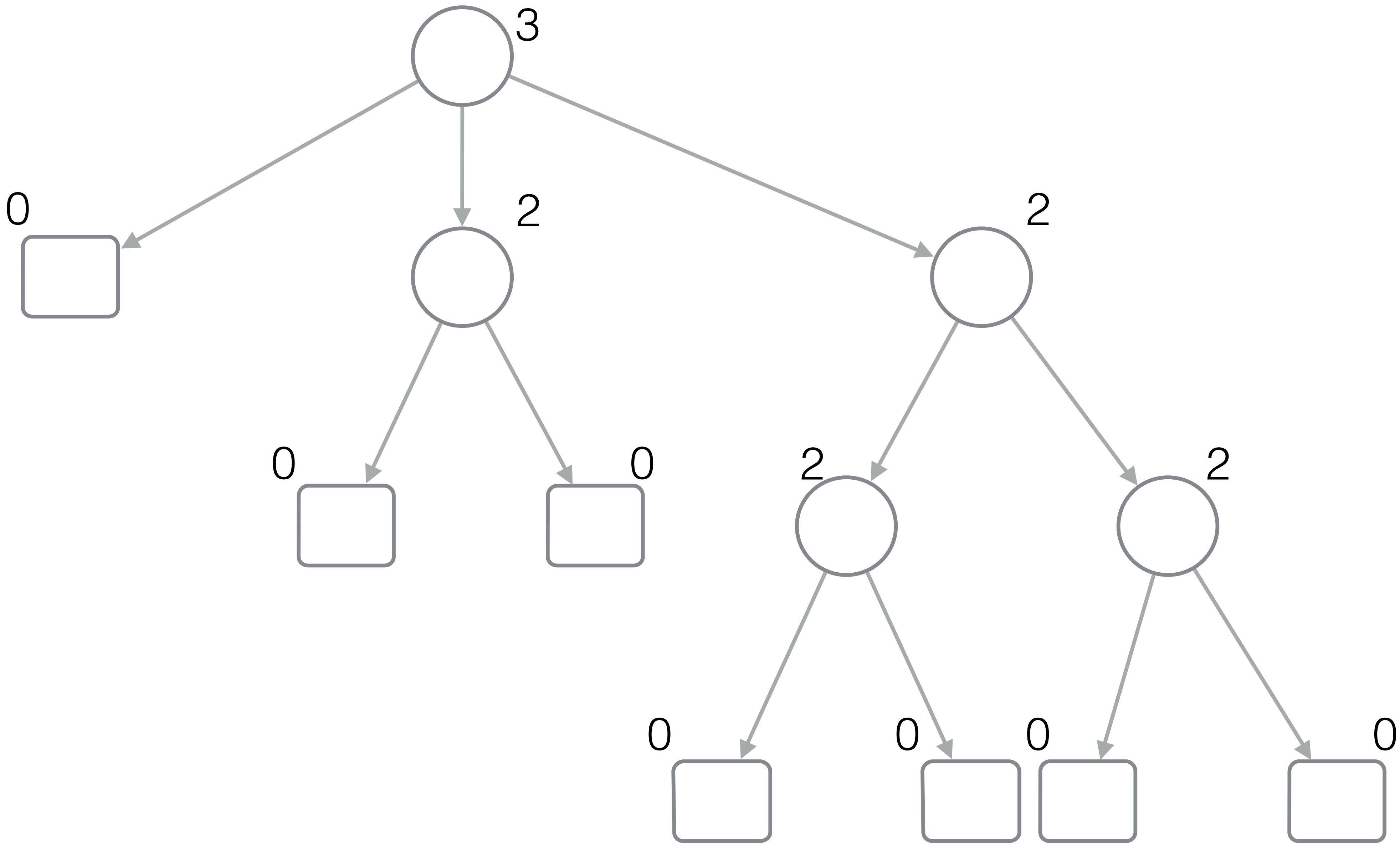


Succinct representation of trees (1)

[LOUDS - Level-order unary degree sequence]

Trivial: O(n log n) bits

Best: 2n bits

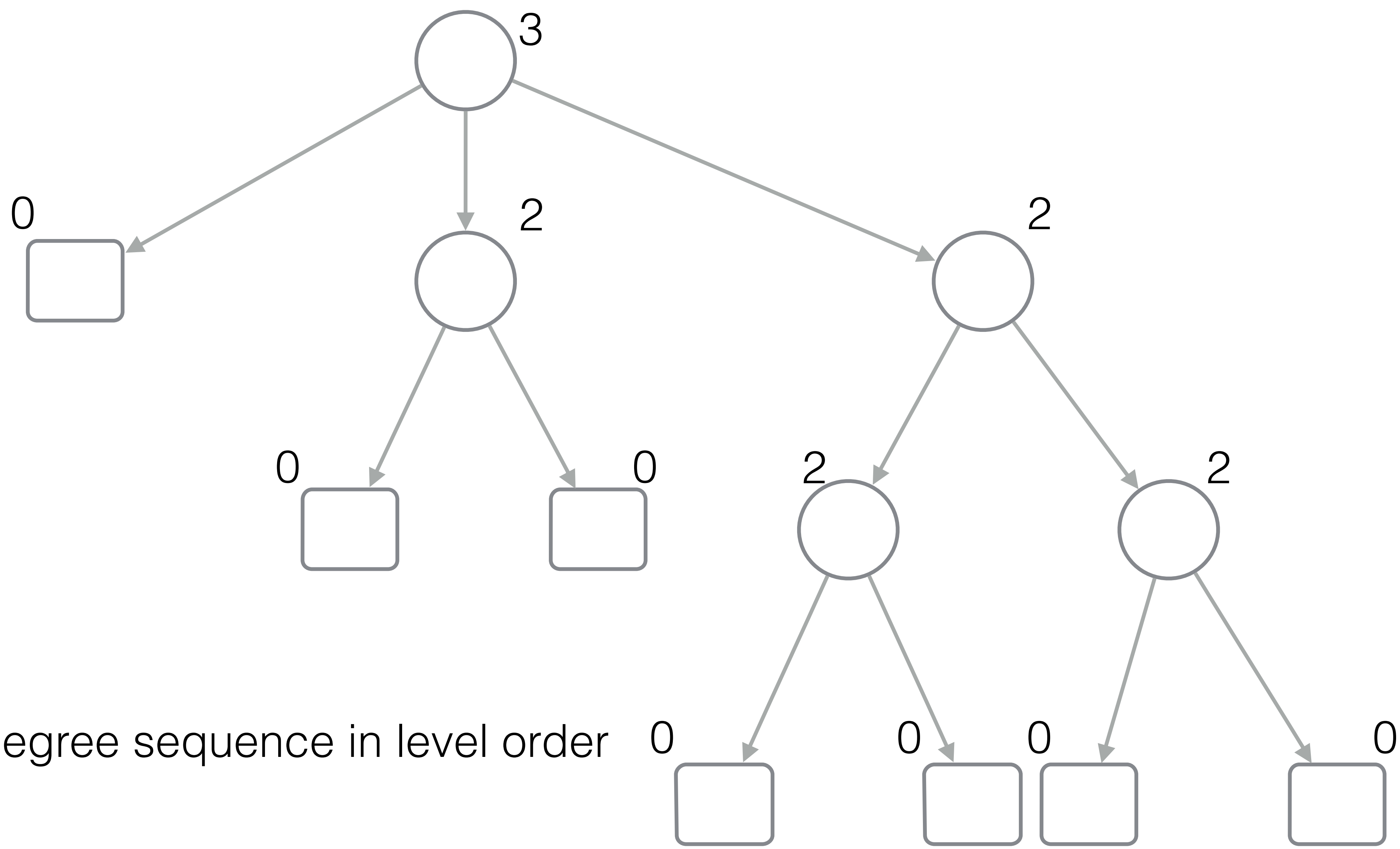


Succinct representation of trees (1)

[LOUDS - Level-order unary degree sequence]

Trivial: O(n log n) bits

Best: 2n bits



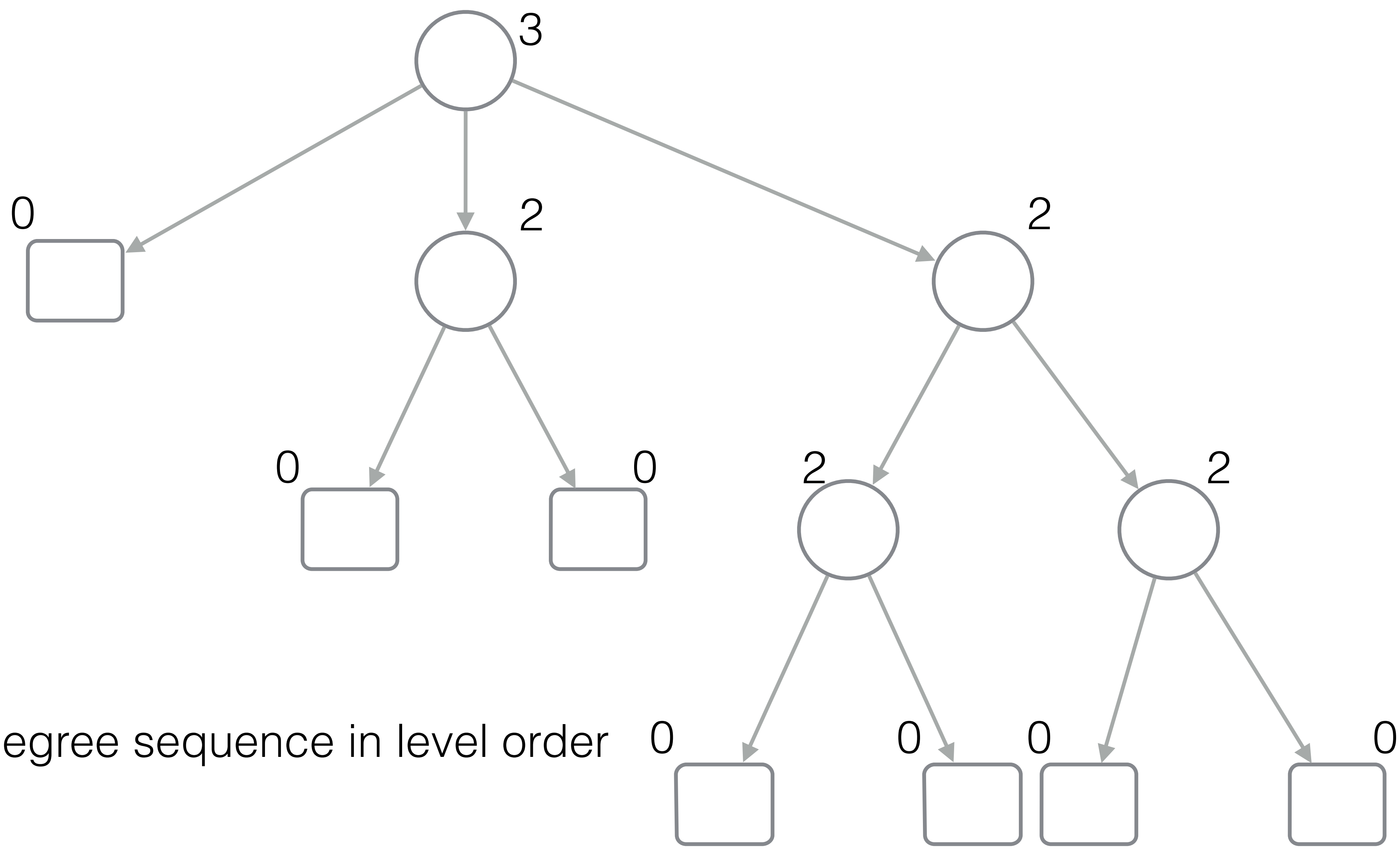
Write the degree sequence in level order

Succinct representation of trees (1)

[LOUDS - Level-order unary degree sequence]

Trivial: O(n log n) bits

Best: 2n bits



Write the degree sequence in level order

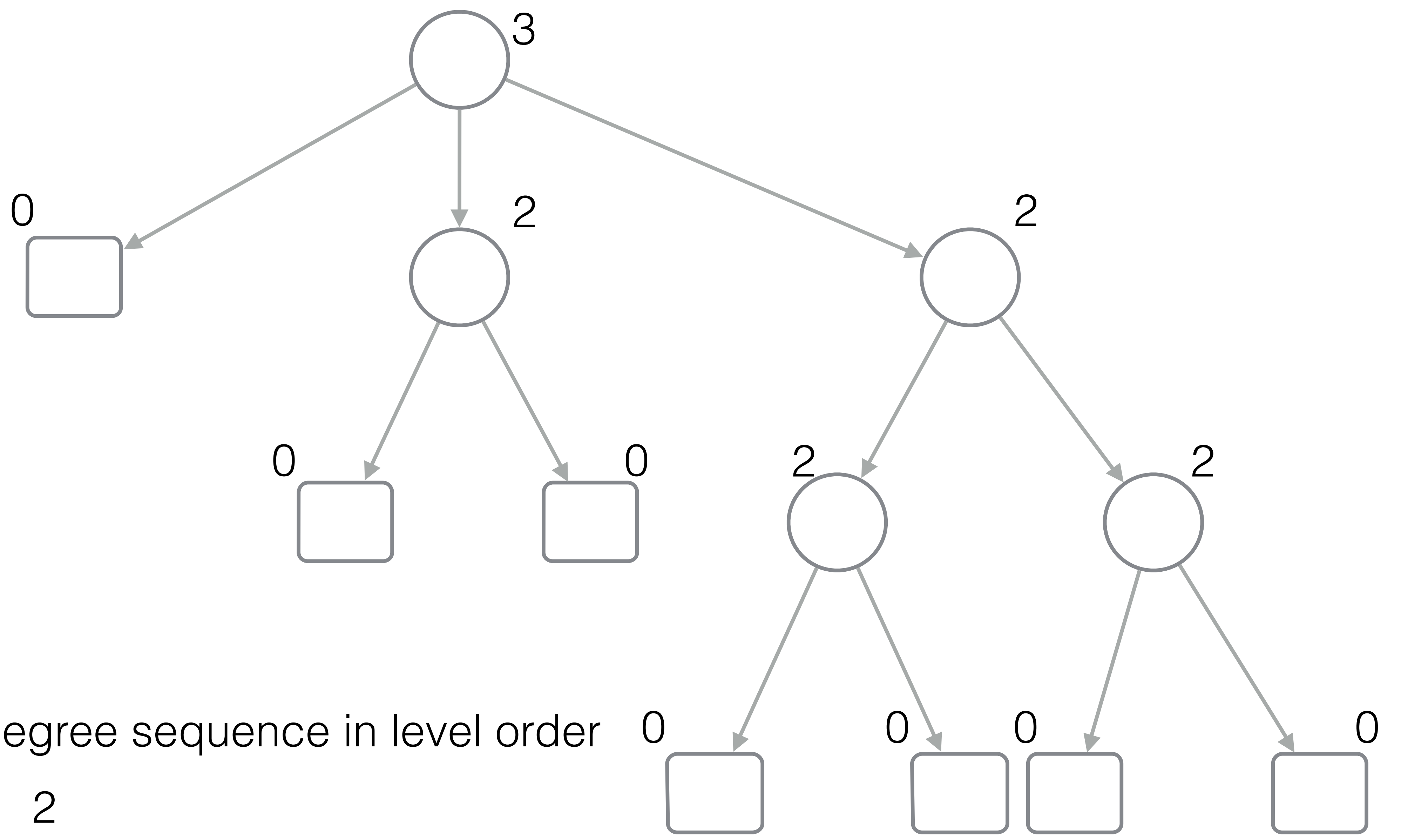
D 3

Succinct representation of trees (1)

[LOUDS - Level-order unary degree sequence]

Trivial: O(n log n) bits

Best: 2n bits



Write the degree sequence in level order

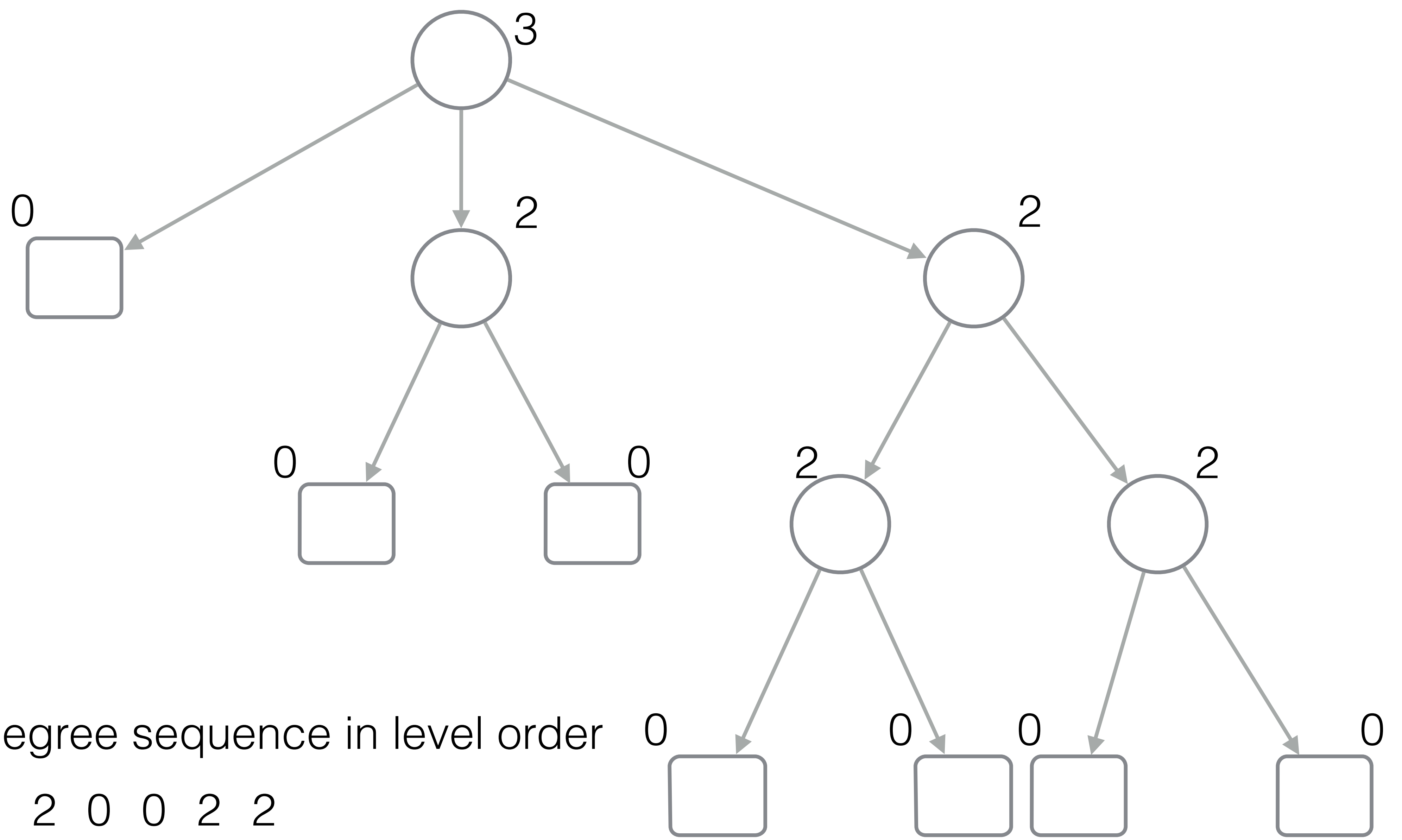
D 3 0 2 2

Succinct representation of trees (1)

[LOUDS - Level-order unary degree sequence]

Trivial: O(n log n) bits

Best: 2n bits



Write the degree sequence in level order

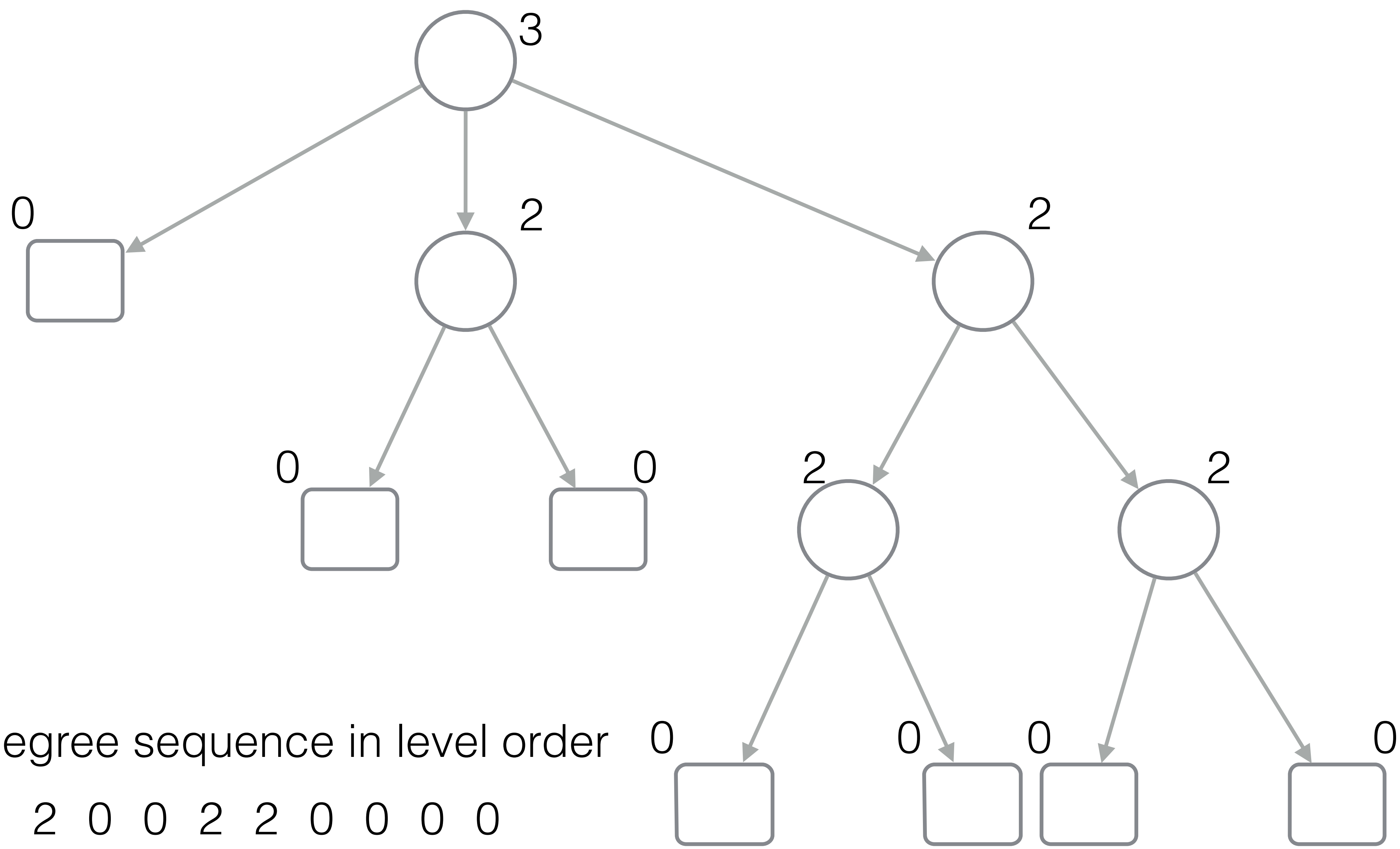
D 3 0 2 2 0 0 2 2

Succinct representation of trees (1)

[LOUDS - Level-order unary degree sequence]

Trivial: O(n log n) bits

Best: 2n bits



Write the degree sequence in level order

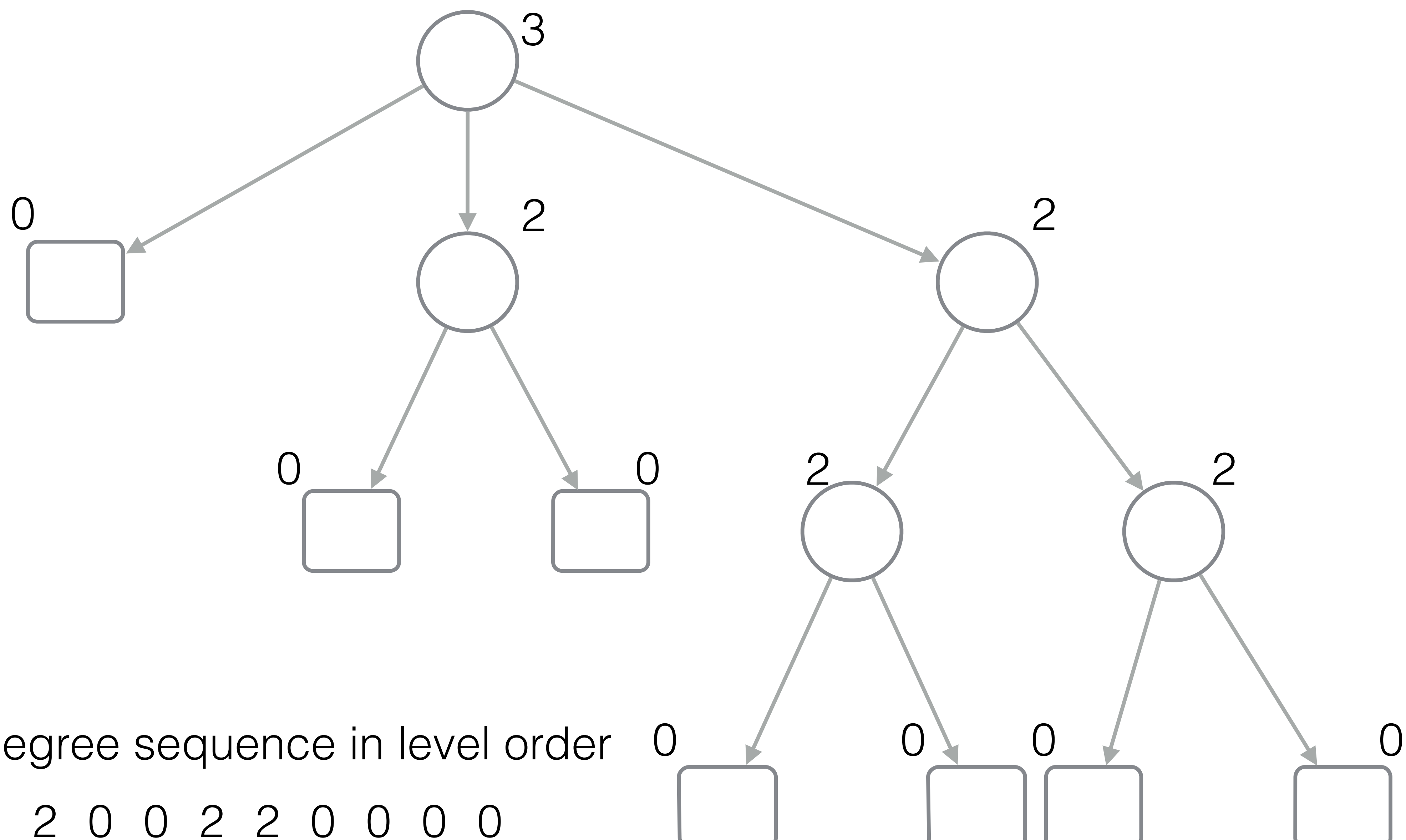
D 3 0 2 2 0 0 2 2 0 0 0 0

Succinct representation of trees (1)

[LOUDS - Level-order unary degree sequence]

Trivial: O(n log n) bits

Best: 2n bits



Write the degree sequence in level order

D 3 0 2 2 0 0 2 2 0 0 0 0

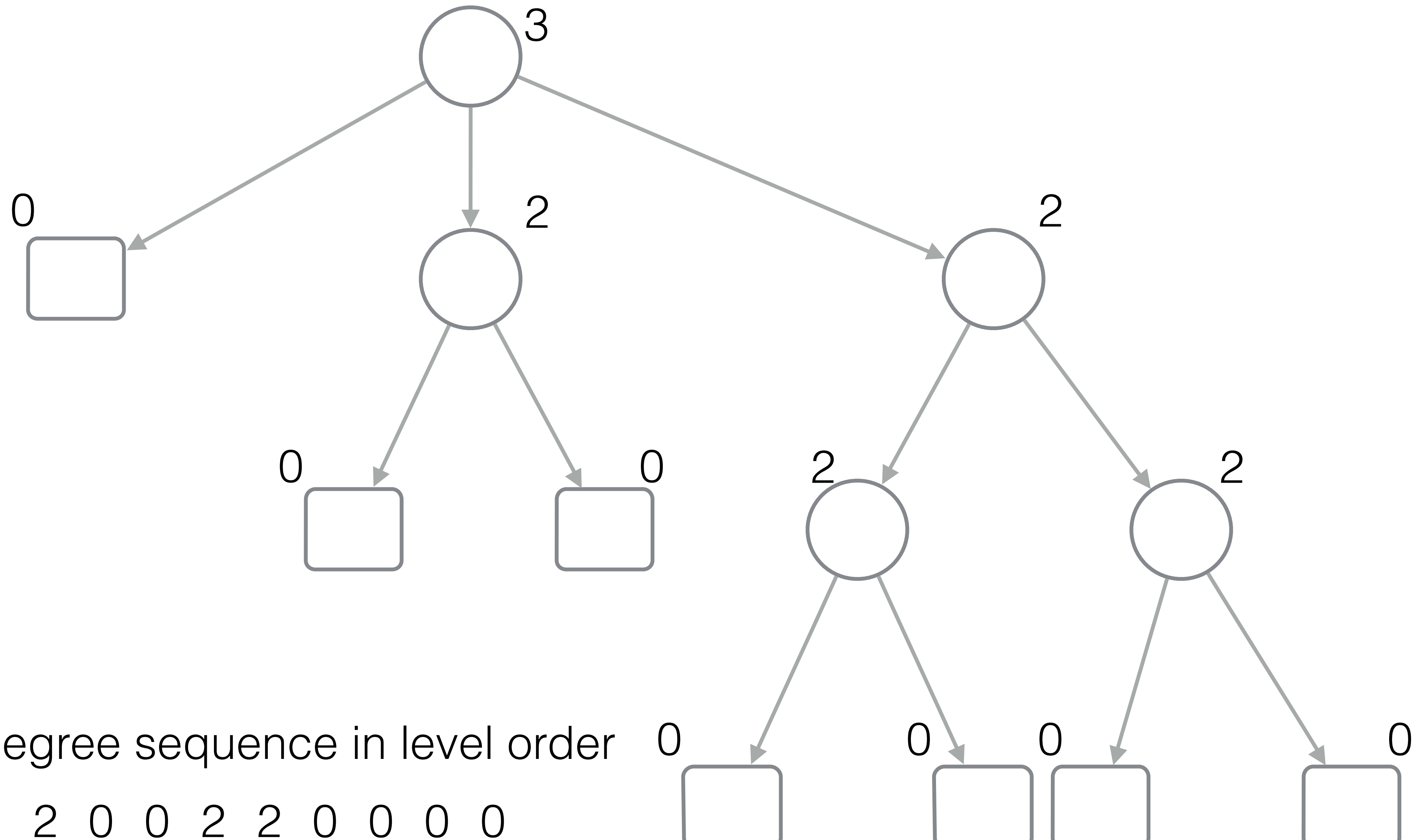
A tree is uniquely determined by the degree sequence

Succinct representation of trees (1)

[LOUDS - Level-order unary degree sequence]

Trivial: O(n log n) bits

Best: 2n bits



Write the degree sequence in level order

D 3 0 2 2 0 0 2 2 0 0 0 0

A tree is uniquely determined by the degree sequence

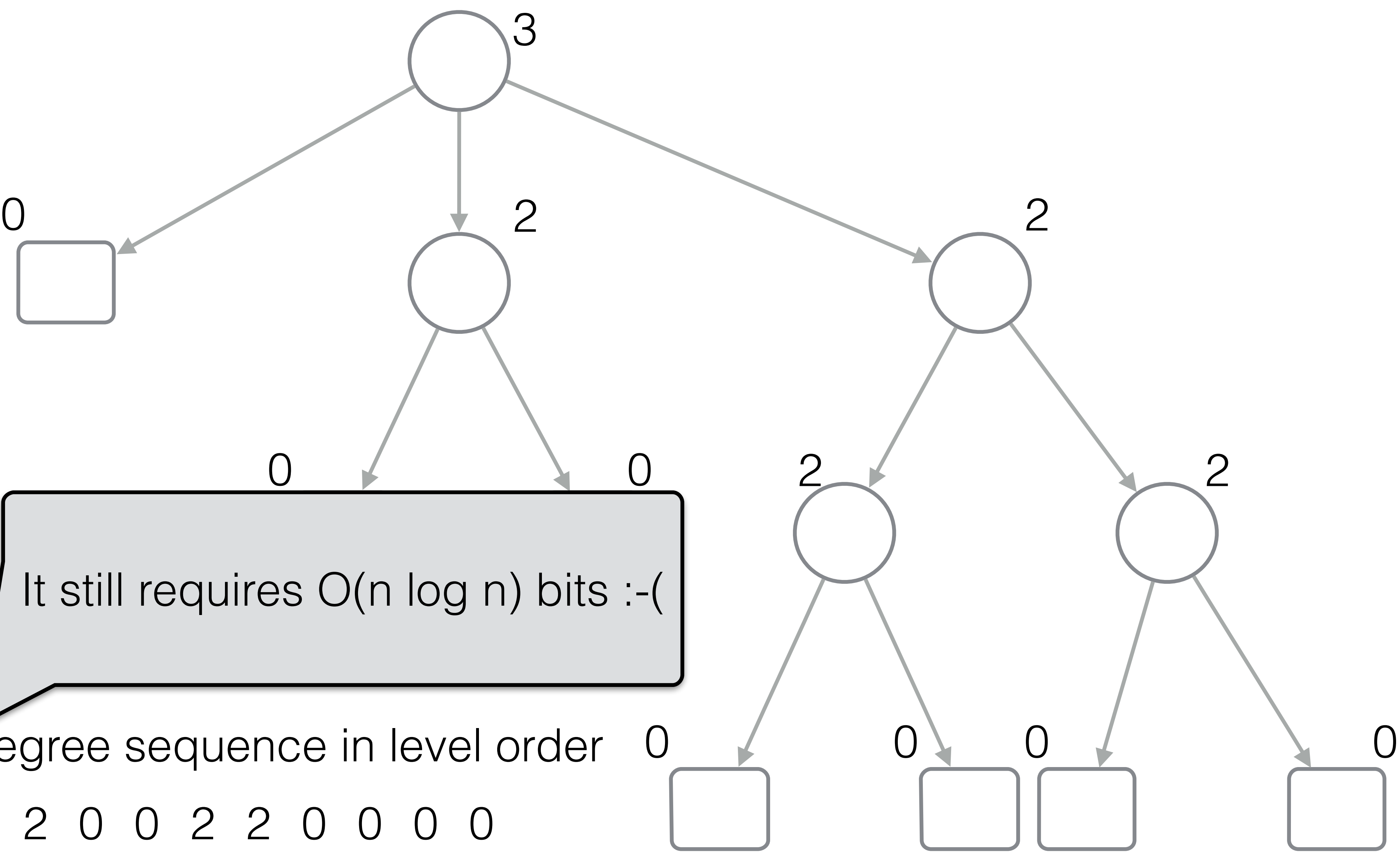
How reconstruct the tree?

Succinct representation of trees (1)

[LOUDS - Level-order unary degree sequence]

Trivial: O(n log n) bits

Best: 2n bits



It still requires O(n log n) bits :-(

Write the degree sequence in level order

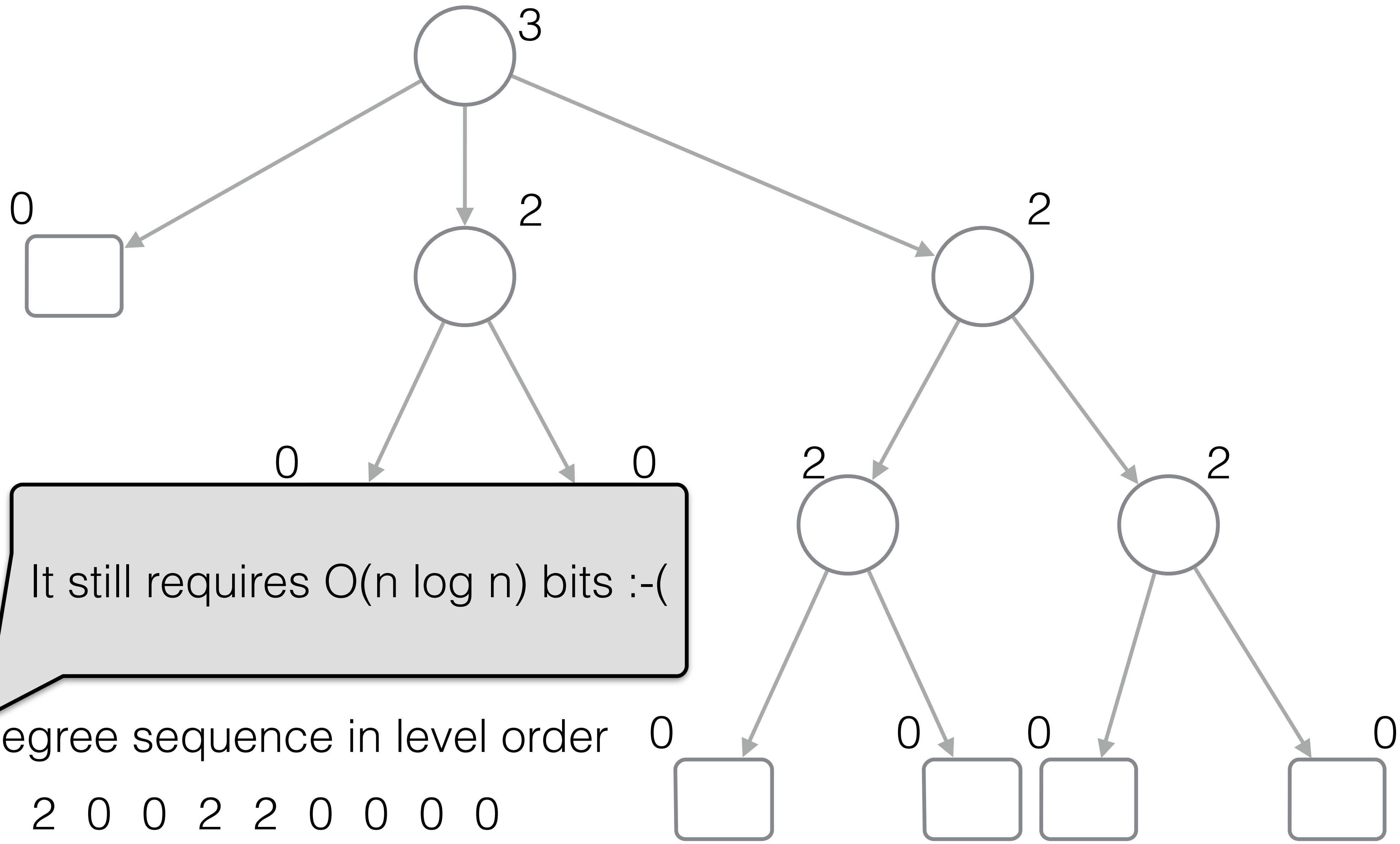
D 3 0 2 2 0 0 2 2 0 0 0 0

Succinct representation of trees (1)

[LOUDS - Level-order unary degree sequence]

Trivial: O(n log n) bits

Best: 2n bits



It still requires O(n log n) bits :-(

Write the degree sequence in level order

D 3 0 2 2 0 0 2 2 0 0 0 0

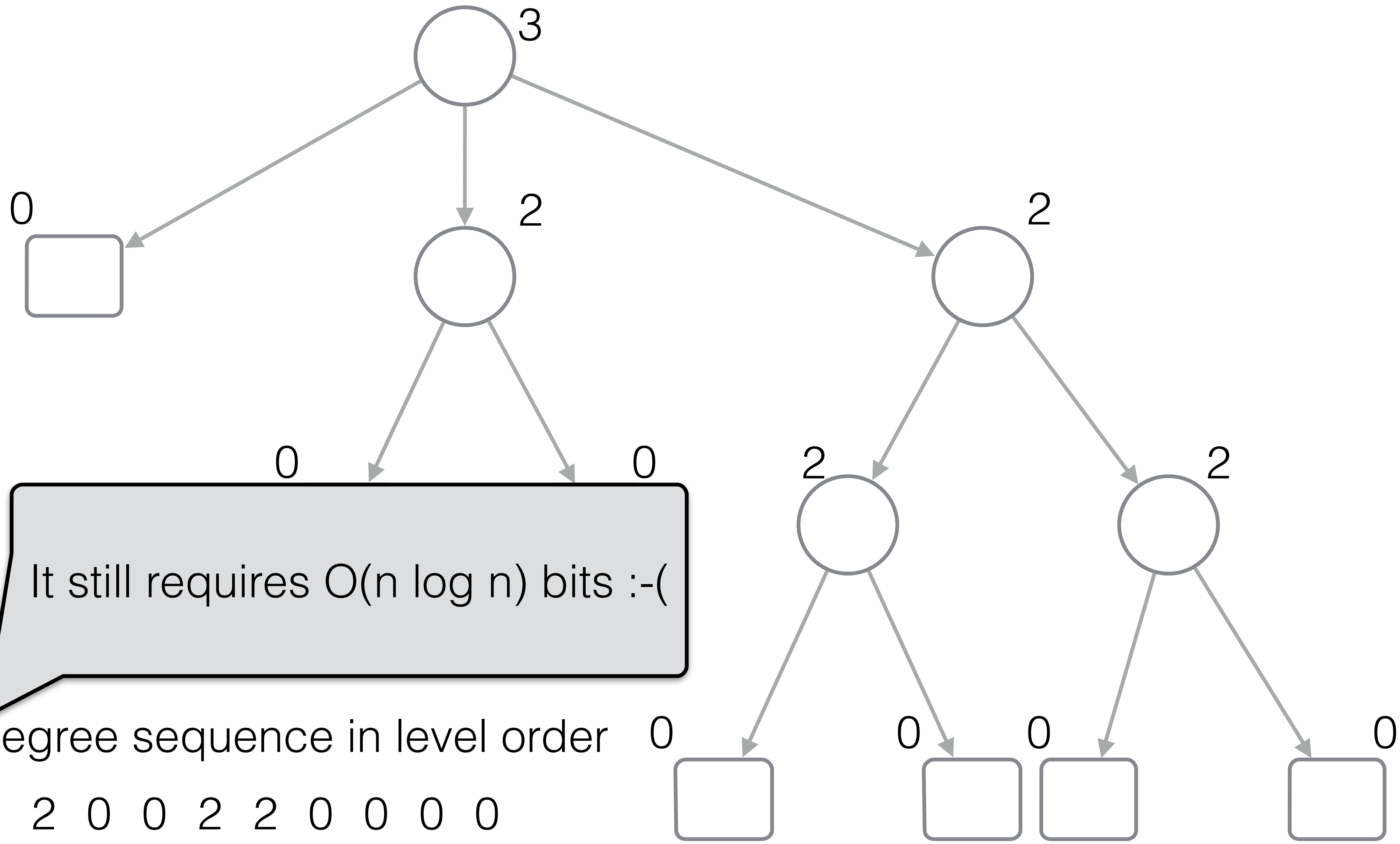
Solution: write them in unary

Succinct representation of trees (1)

[LOUDS - Level-order unary degree sequence]

Trivial: O(n log n) bits

Best: 2n bits



It still requires O(n log n) bits :-(

Write the degree sequence in level order

D 3 0 2 2 0 0 2 2 0 0 0 0

Solution: write them in unary

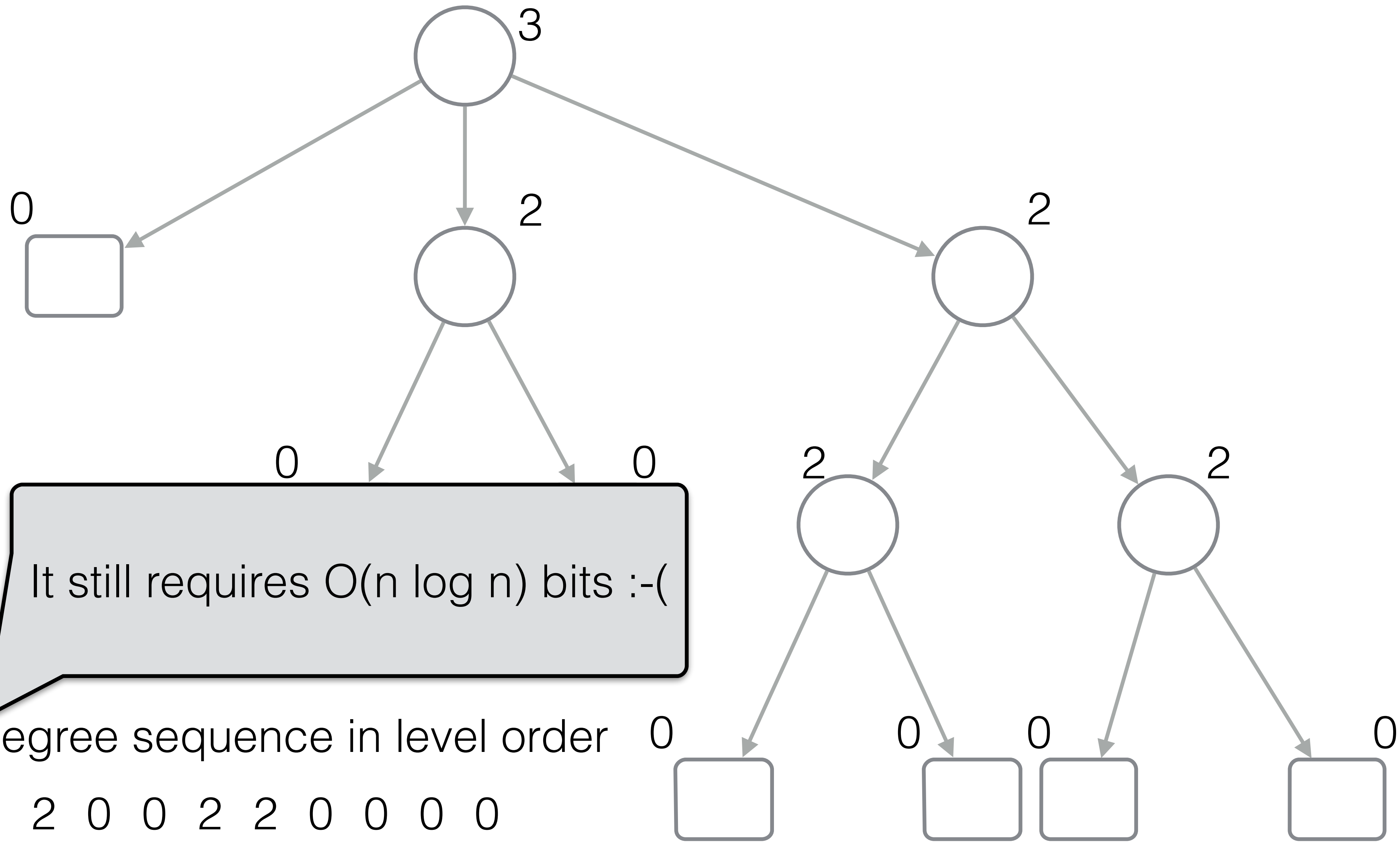
B

Succinct representation of trees (1)

[LOUDS - Level-order unary degree sequence]

Trivial: O(n log n) bits

Best: 2n bits



It still requires O(n log n) bits :-(

Write the degree sequence in level order

D 3 0 2 2 0 0 2 2 0 0 0 0

Solution: write them in unary

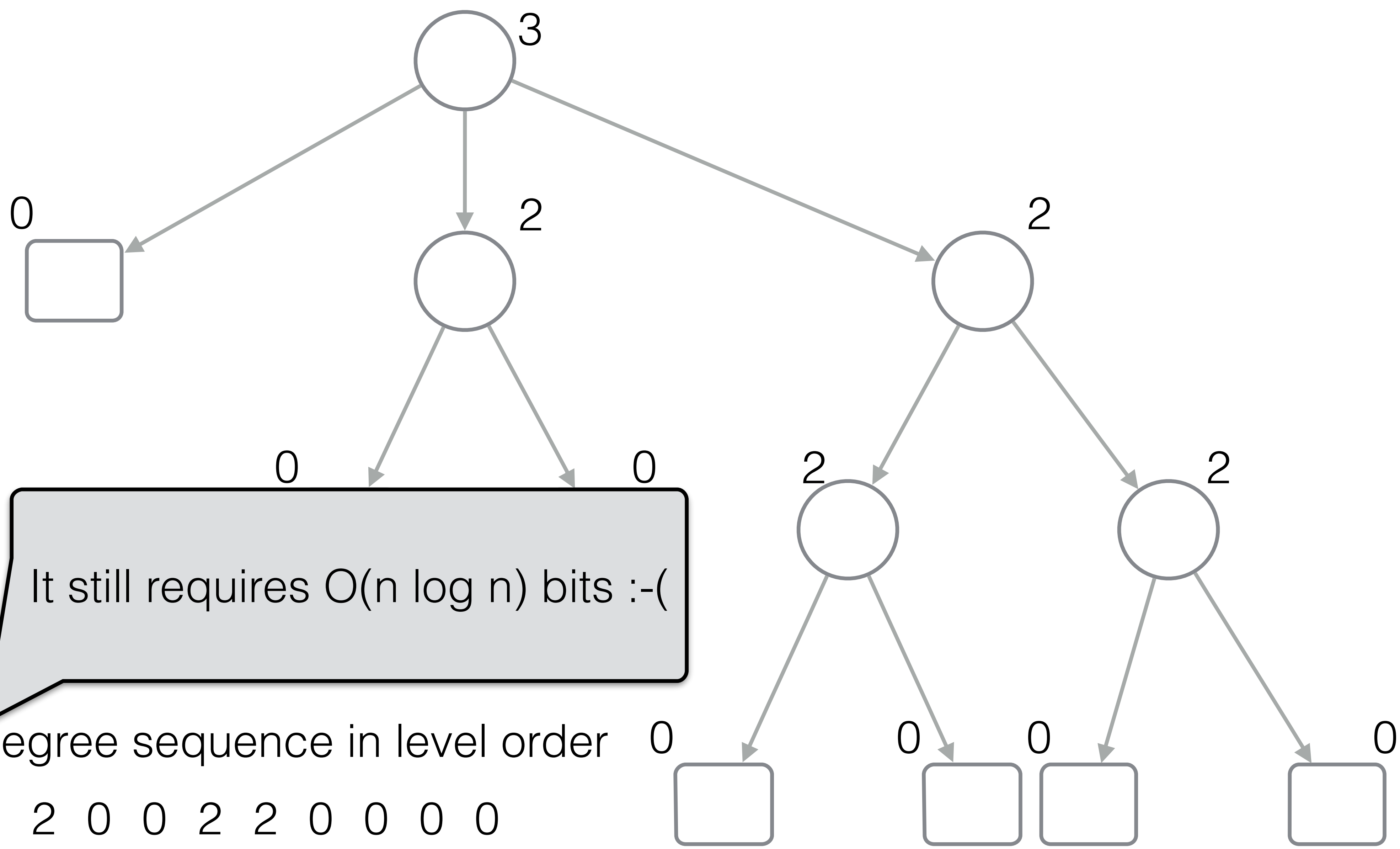
B 1110

Succinct representation of trees (1)

[LOUDS - Level-order unary degree sequence]

Trivial: O(n log n) bits

Best: 2n bits



It still requires O(n log n) bits :-(

Write the degree sequence in level order

D 3 0 2 2 0 0 2 2 0 0 0 0

Solution: write them in unary

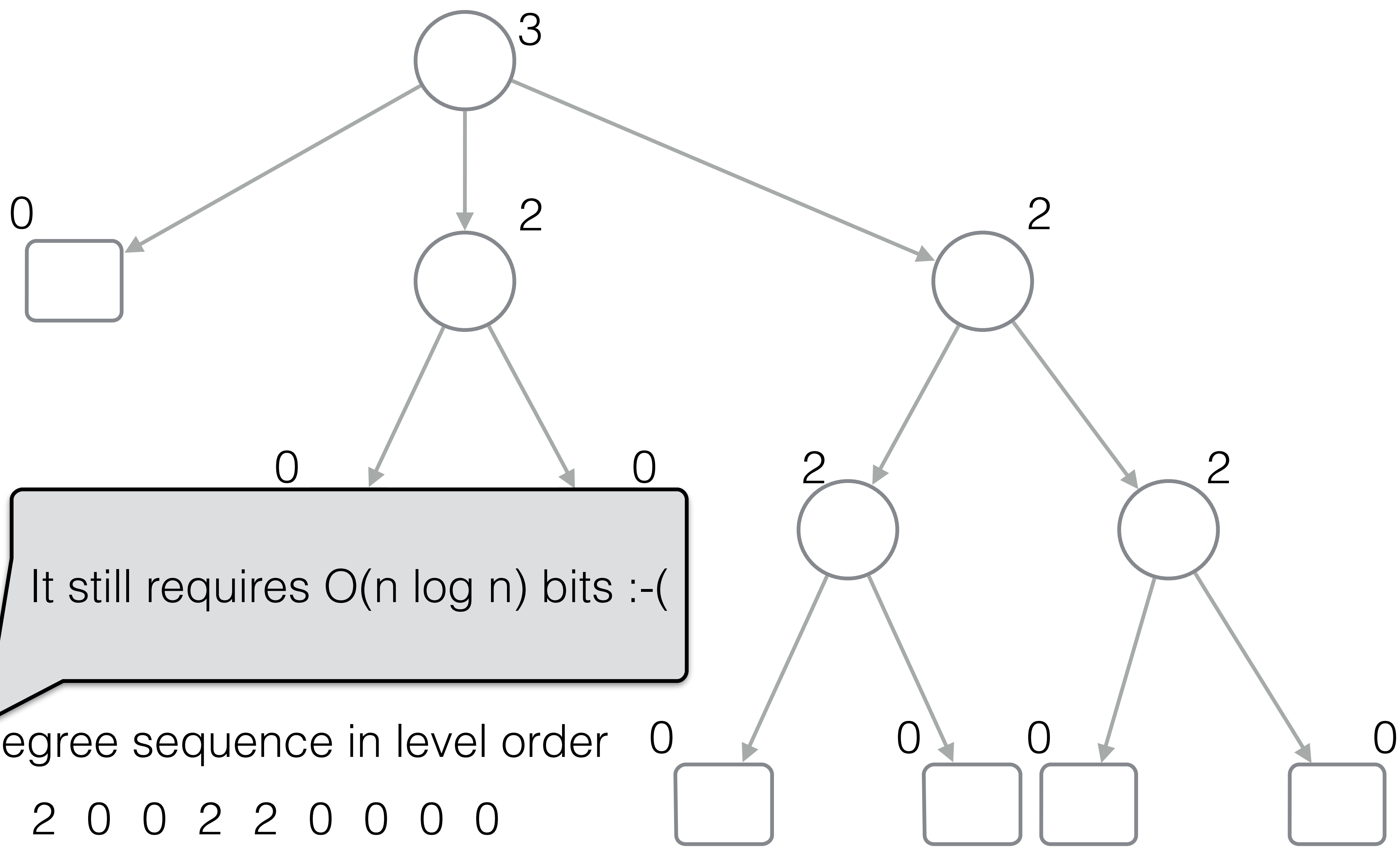
B 1110 0 110 110 0 0 110 110 0 0 0 0

Succinct representation of trees (1)

[LOUDS - Level-order unary degree sequence]

Trivial: O(n log n) bits

Best: 2n bits



It still requires O(n log n) bits :-(

Write the degree sequence in level order

D 3 0 2 2 0 0 2 2 0 0 0 0

Solution: write them in unary

B 1110 0 110 110 0 0 110 110 0 0 0 0

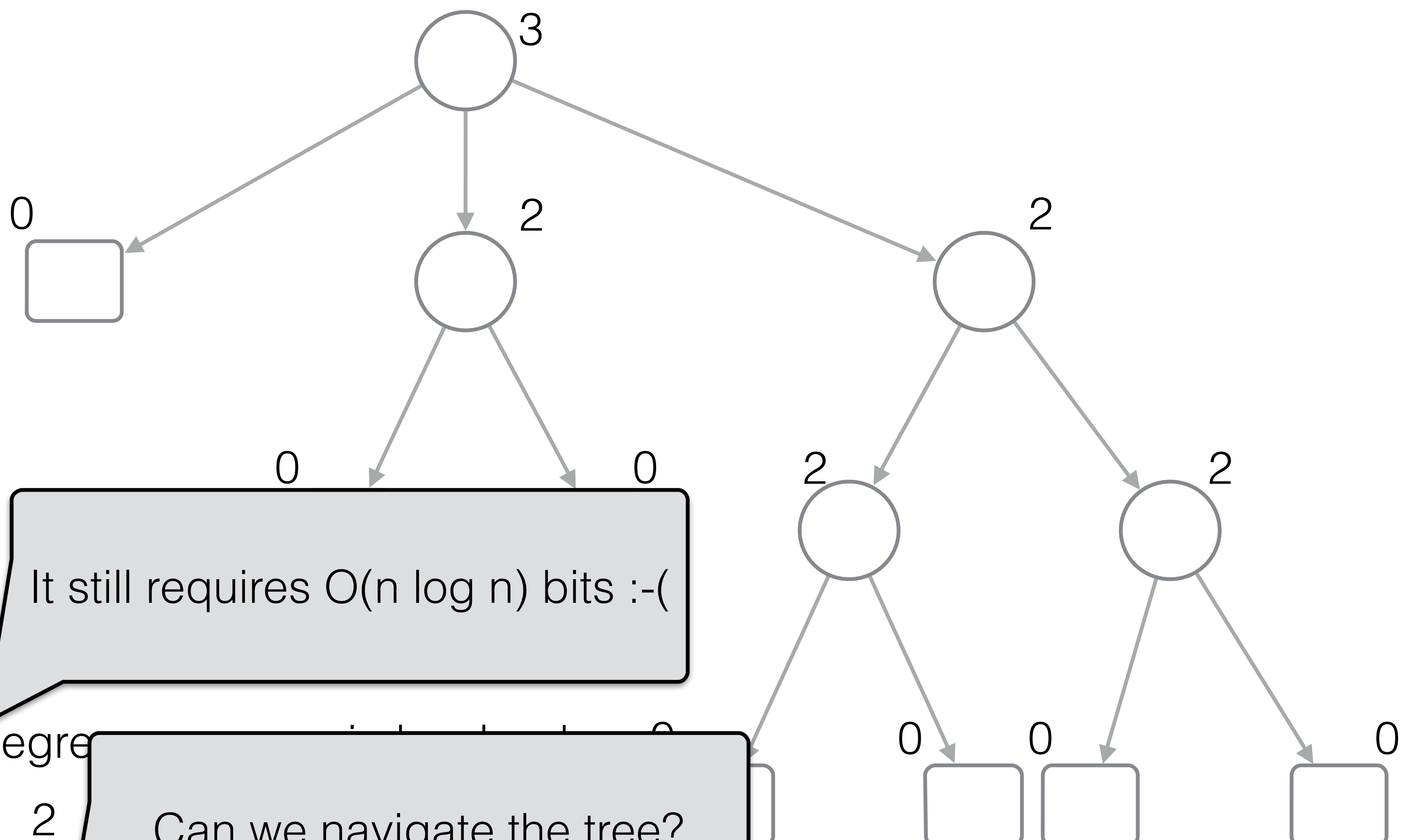
B takes 2n - 1 bits!
For each node we have a 0 and a 1
(but the root)

Succinct representation of trees (1)

[LOUDS - Level-order unary degree sequence]

Trivial: O(n log n) bits

Best: 2n bits



It still requires O(n log n) bits :-(

Write the degre

D 3 0 2 2

Can we navigate the tree?

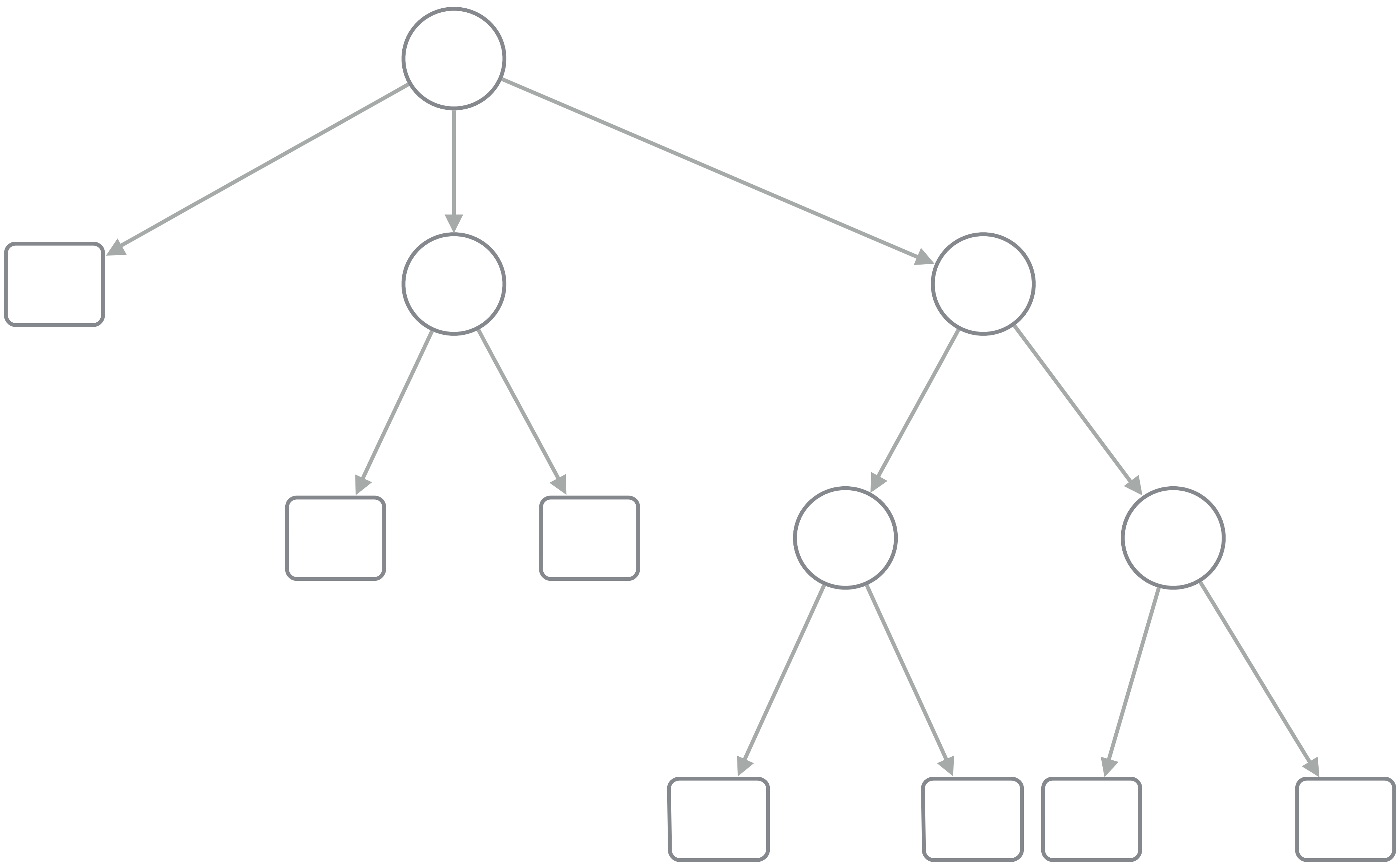
Solution: write ...em in unary

B 1110 0 110 110 0 0 110 110 0 0 0 0

B takes 2n - 1 bits!
For each node we have a 0 and a 1
(but the root)

Succinct representation of trees (1)

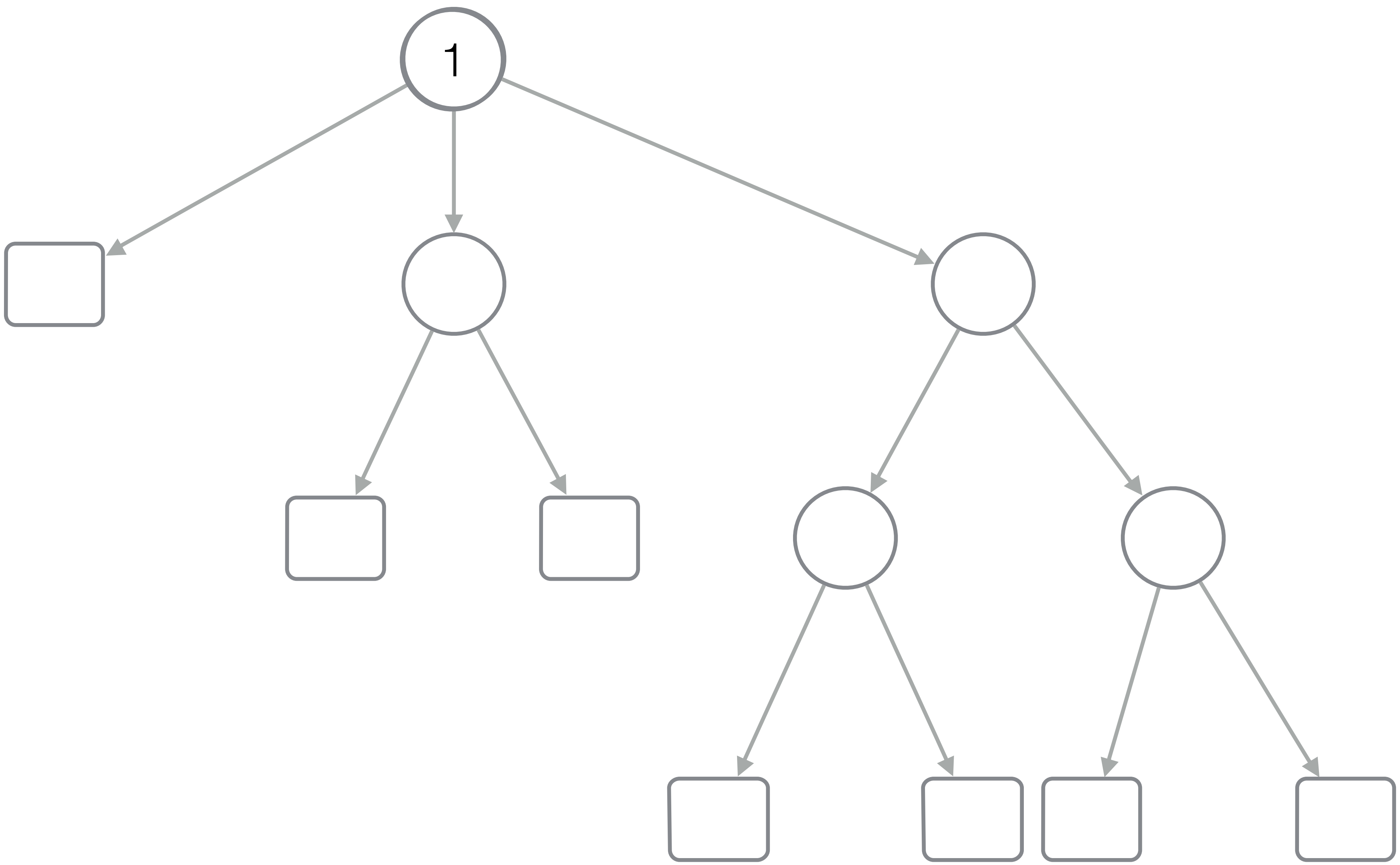
[LOUDS - Level-order unary degree sequence]



B 1 1 1 0 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 0 0

Succinct representation of trees (1)

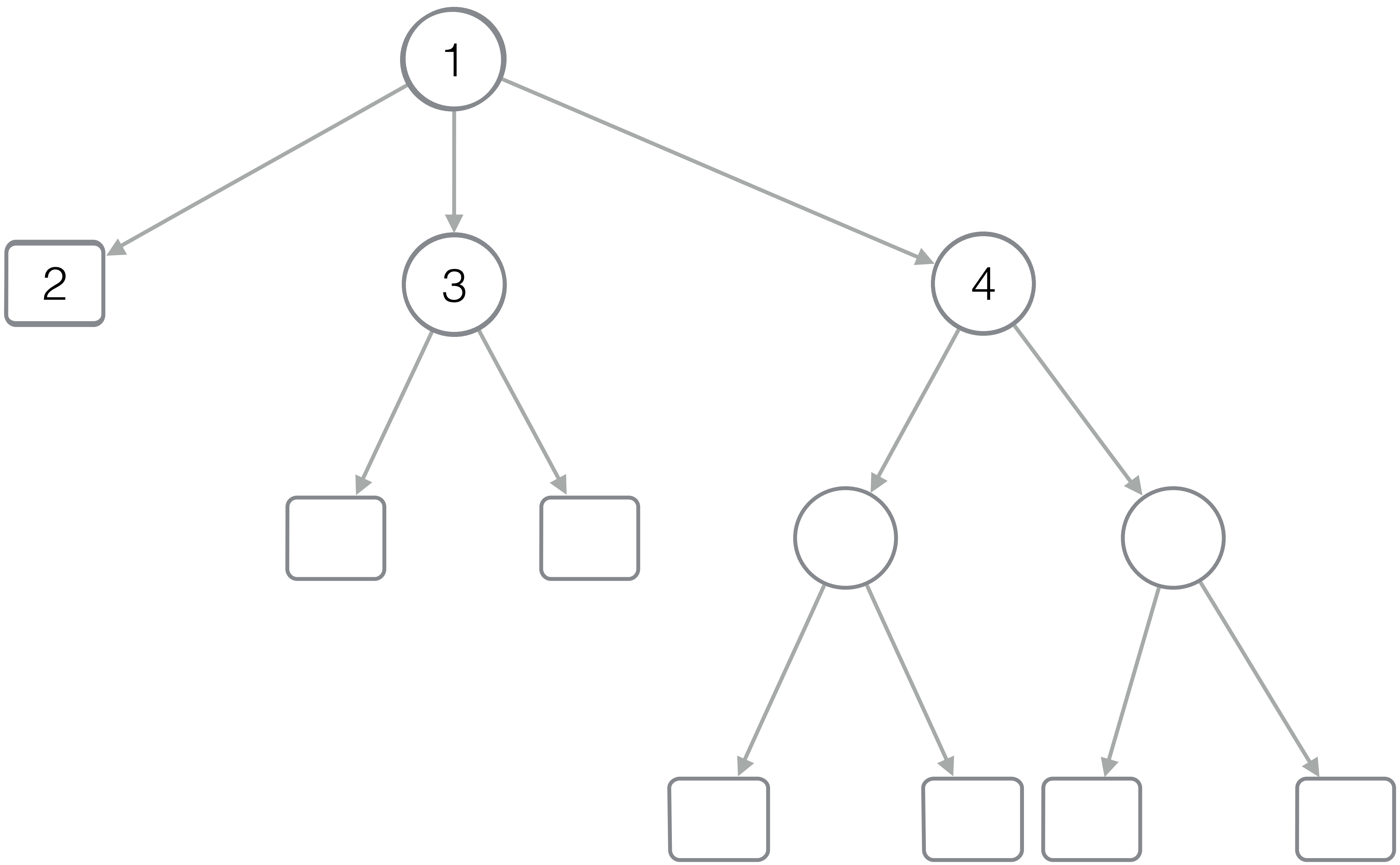
[LOUDS - Level-order unary degree sequence]



B 1 1 1 0 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 0 0

Succinct representation of trees (1)

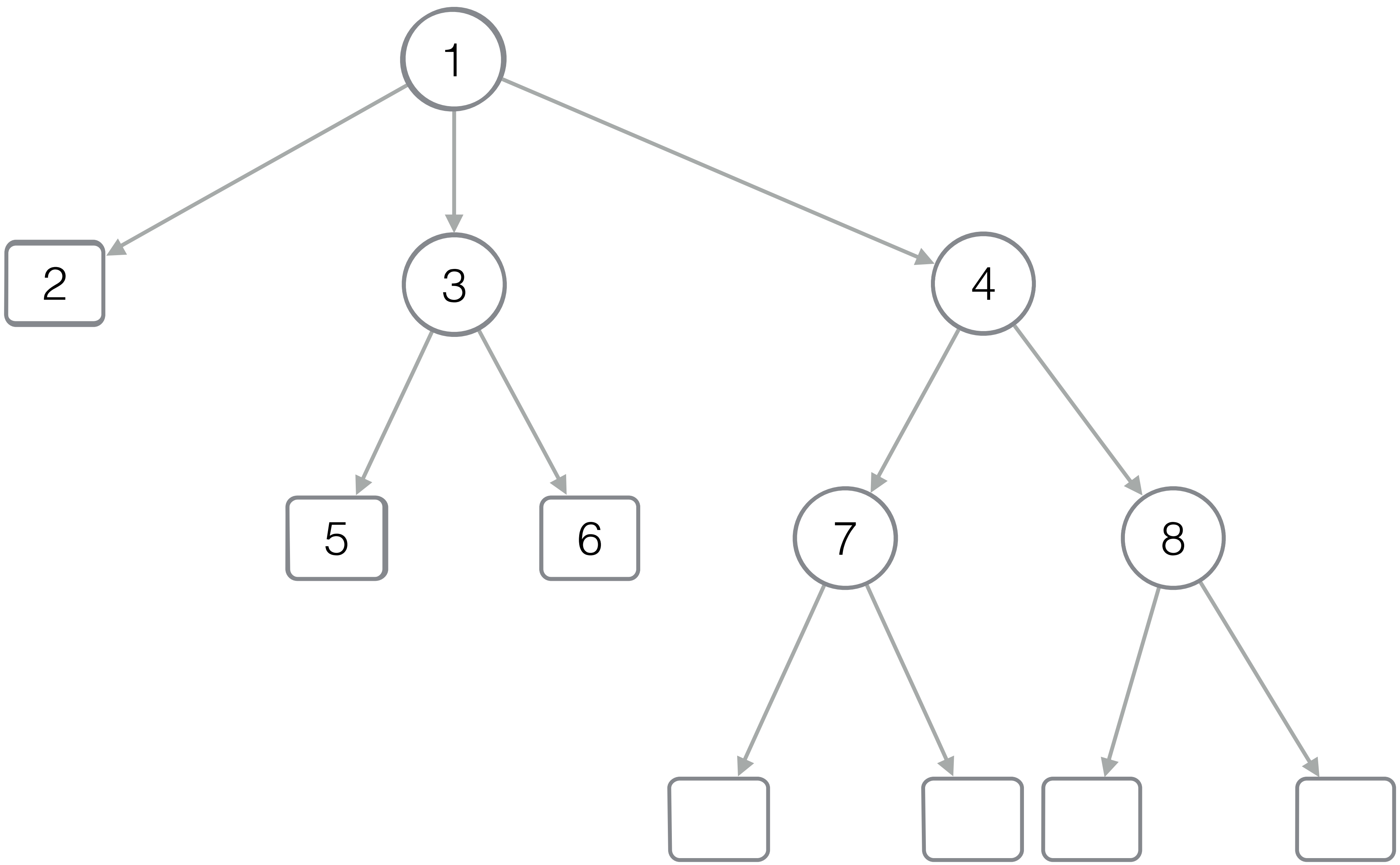
[LOUDS - Level-order unary degree sequence]



B 1 1 1 0 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 0 0

Succinct representation of trees (1)

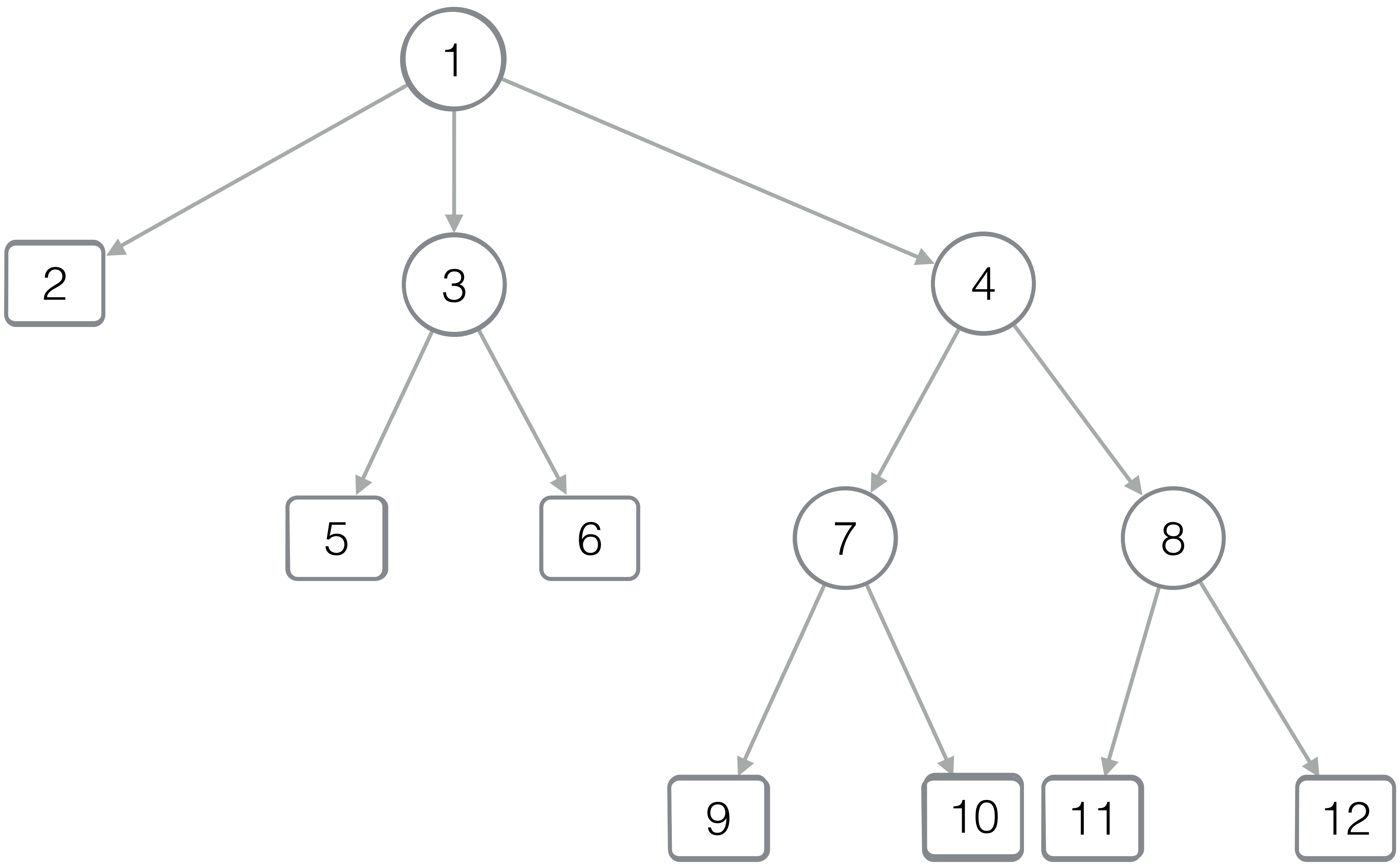
[LOUDS - Level-order unary degree sequence]



B 1 1 1 0 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 0 0

Succinct representation of trees (1)

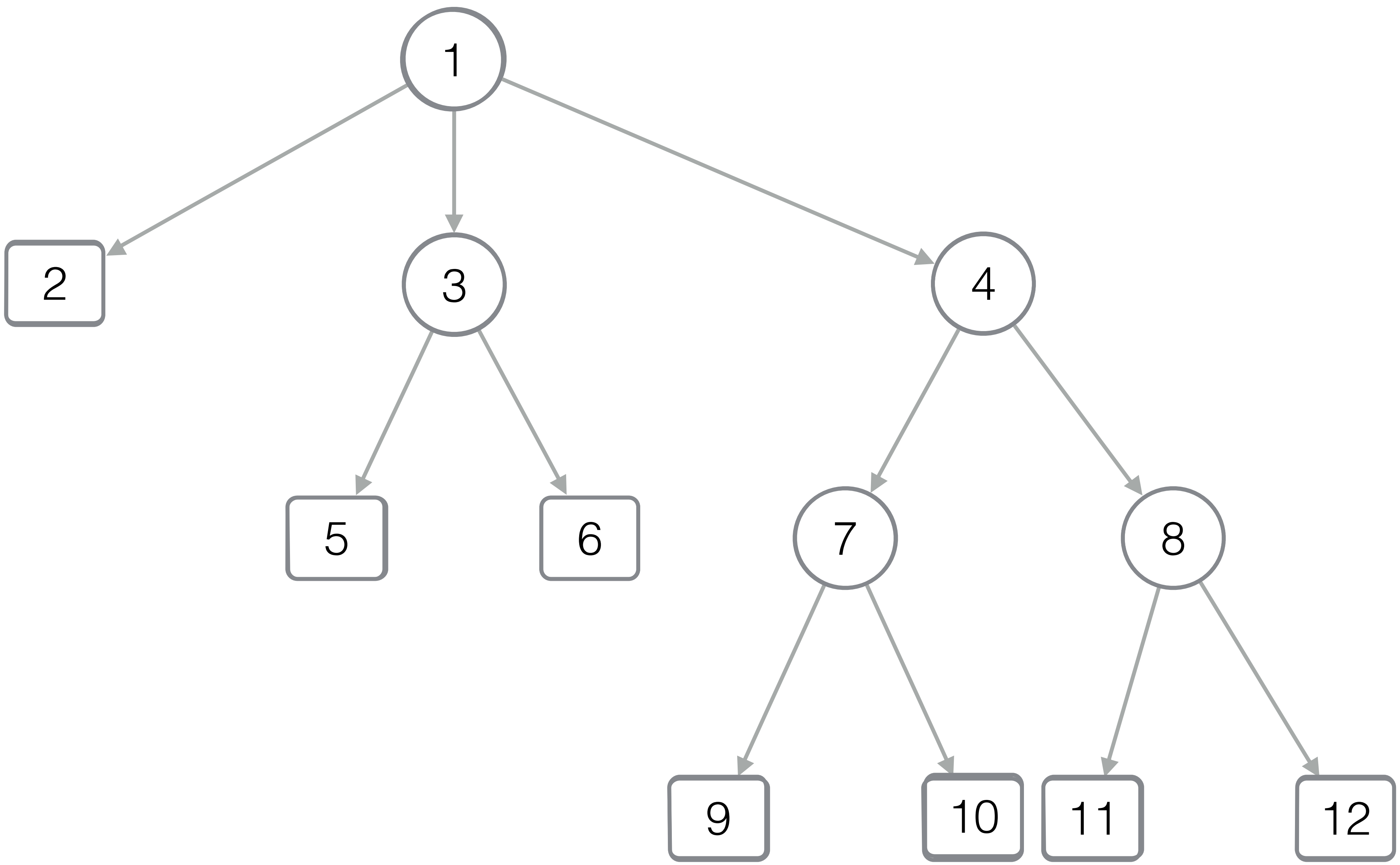
[LOUDS - Level-order unary degree sequence]



B 1 1 1 0 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 0 0

Succinct representation of trees (1)

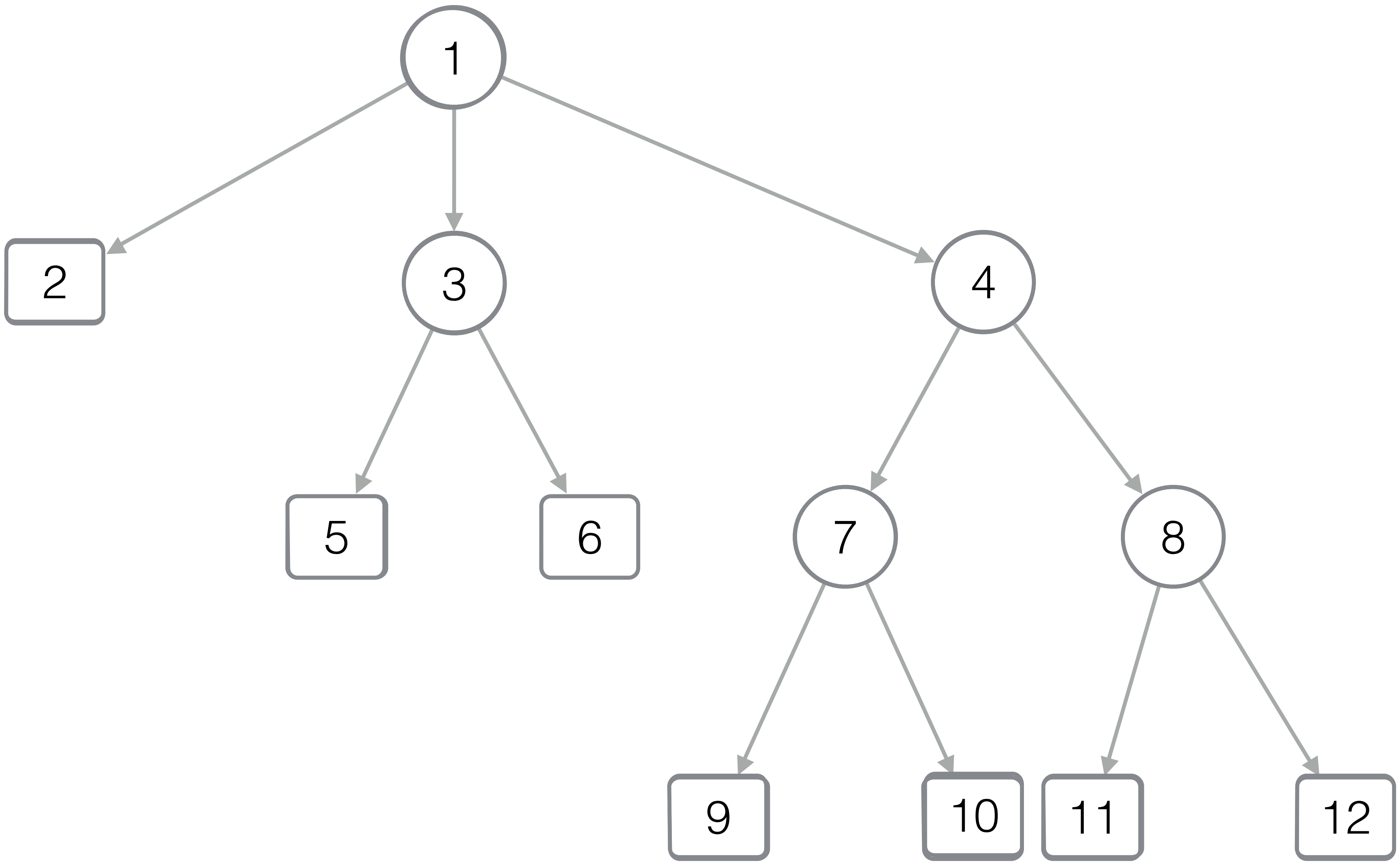
[LOUDS - Level-order unary degree sequence]



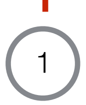
B 1 0 1 1 1 0 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 0 0

Succinct representation of trees (1)

[LOUDS - Level-order unary degree sequence]

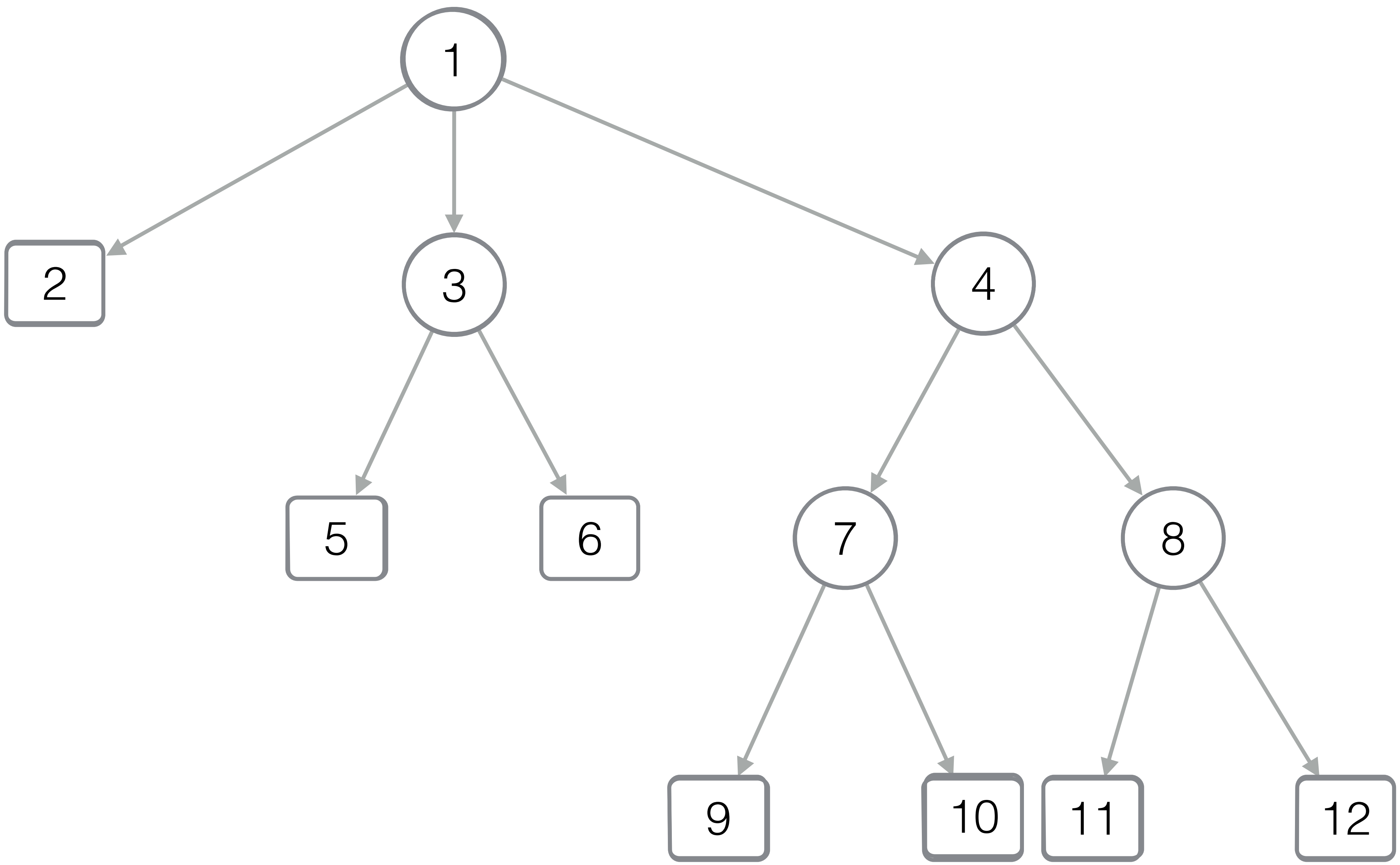


B 1 0 1 1 1 0 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 0 0



Succinct representation of trees (1)

[LOUDS - Level-order unary degree sequence]



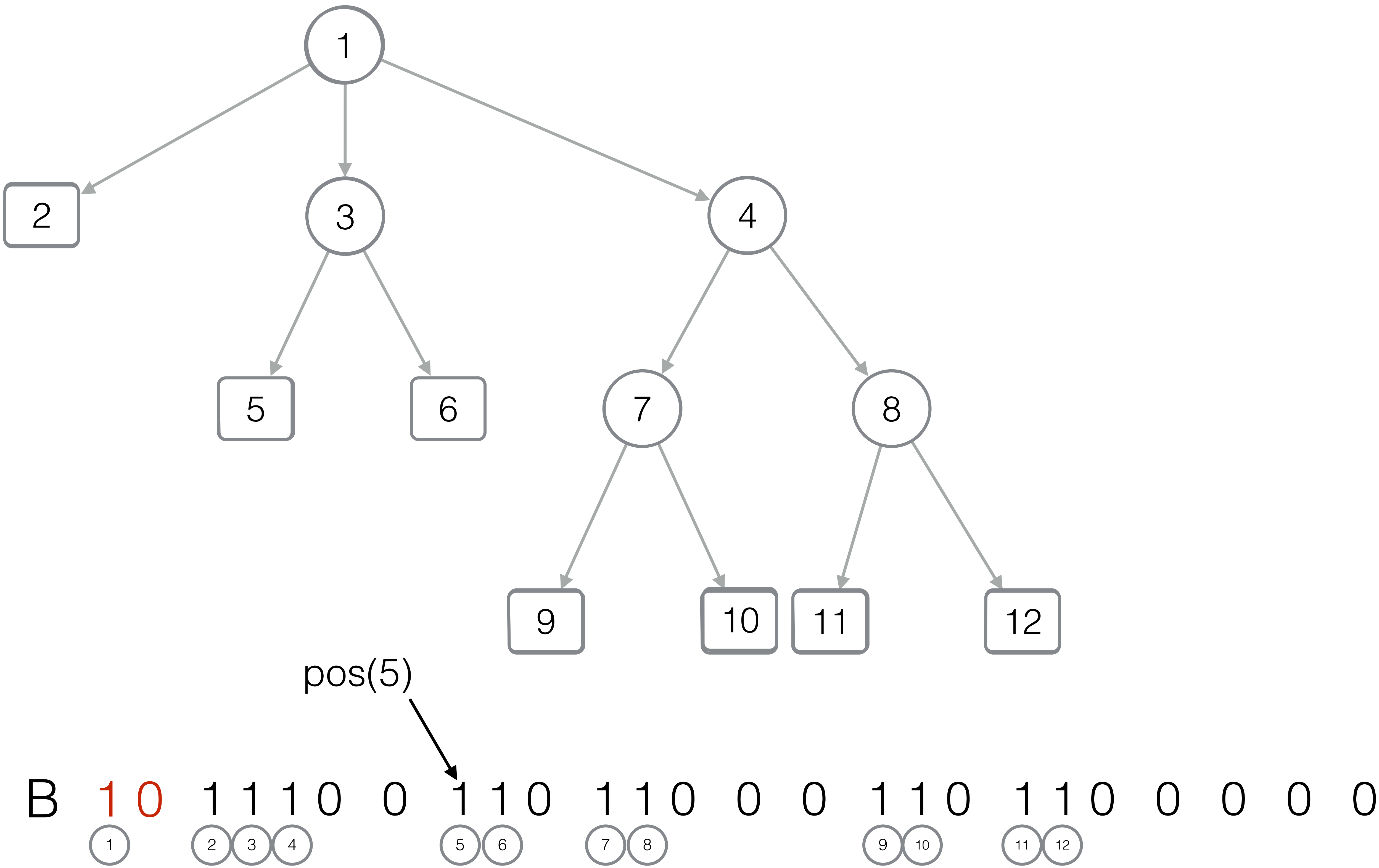
B 1 0 1 1 1 0 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 0 0

1 2 3 4 5 6 7 8 9 10 11 12

Succinct representation of trees (1)

[LOUDS - Level-order unary degree sequence]

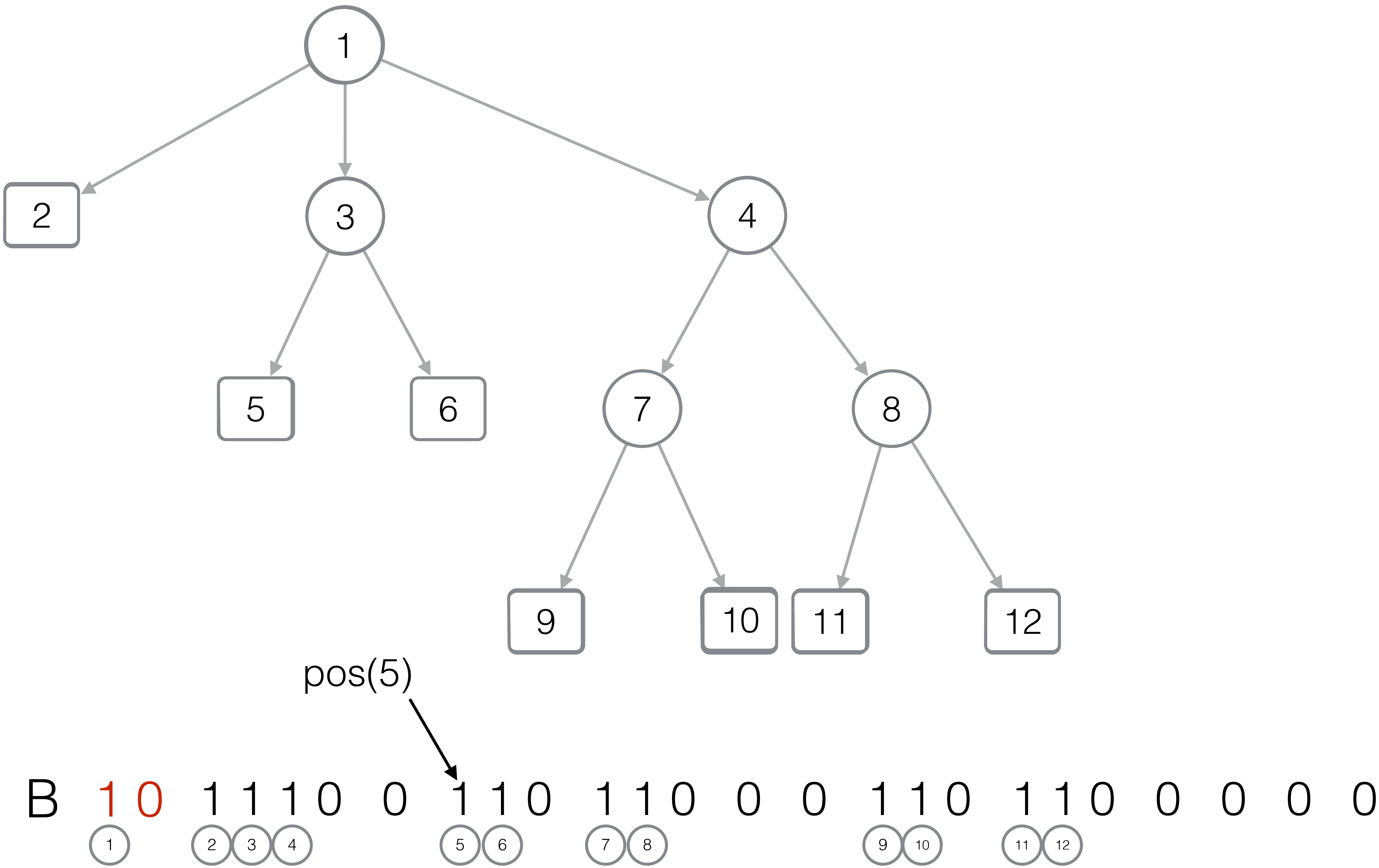
pos(x) =



Succinct representation of trees (1)

[LOUDS - Level-order unary degree sequence]

$pos(x) = Select_1(x)$

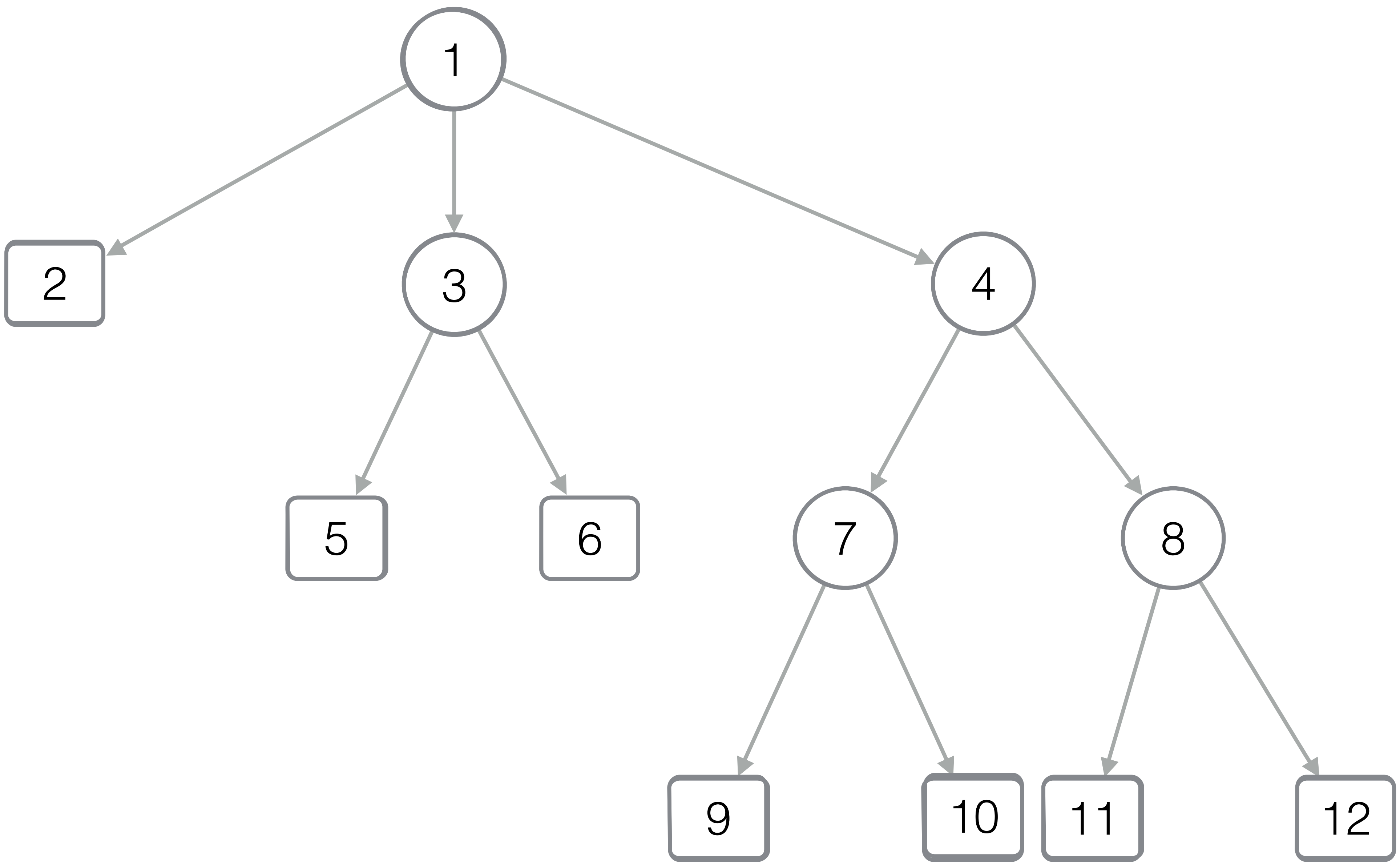


Succinct representation of trees (1)

[LOUDS - Level-order unary degree sequence]

$pos(x) = Select_1(x)$

$firstChild(x) = ?$



B 1 0 1 1 1 0 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 0 0

(1) (2) (3) (4) (5) (6) (7) (8) (9) (10) (11) (12)

Succinct representation of trees (1)

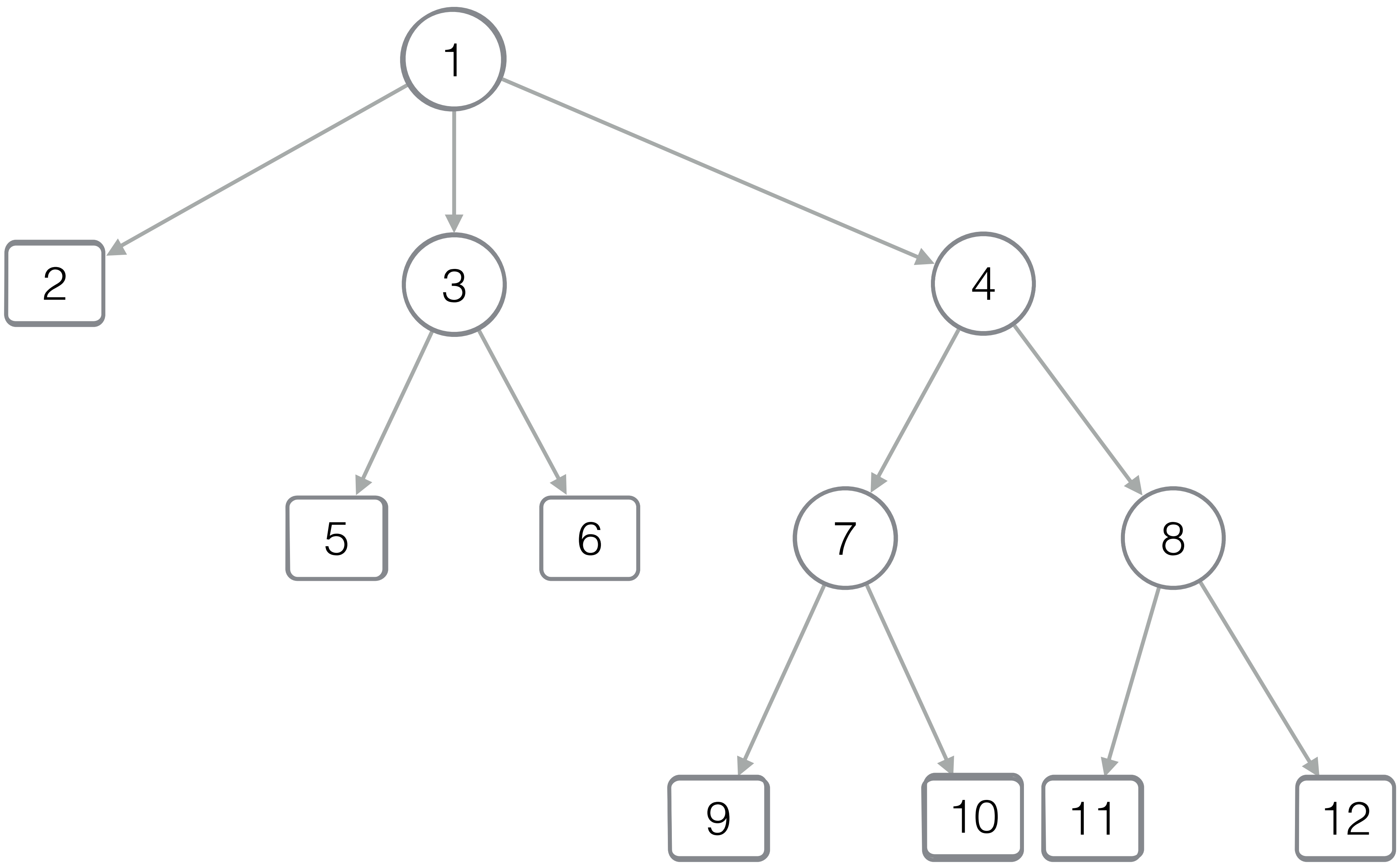
[LOUDS - Level-order unary degree sequence]

$pos(x) = Select_1(x)$

$firstChild(x) = ?$

$y = Select_0(x)+1$

// start of x's children in B



B 1 0 1 1 1 0 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 0 0

1 2 3 4 5 6 7 8 9 10 11 12

Succinct representation of trees (1)

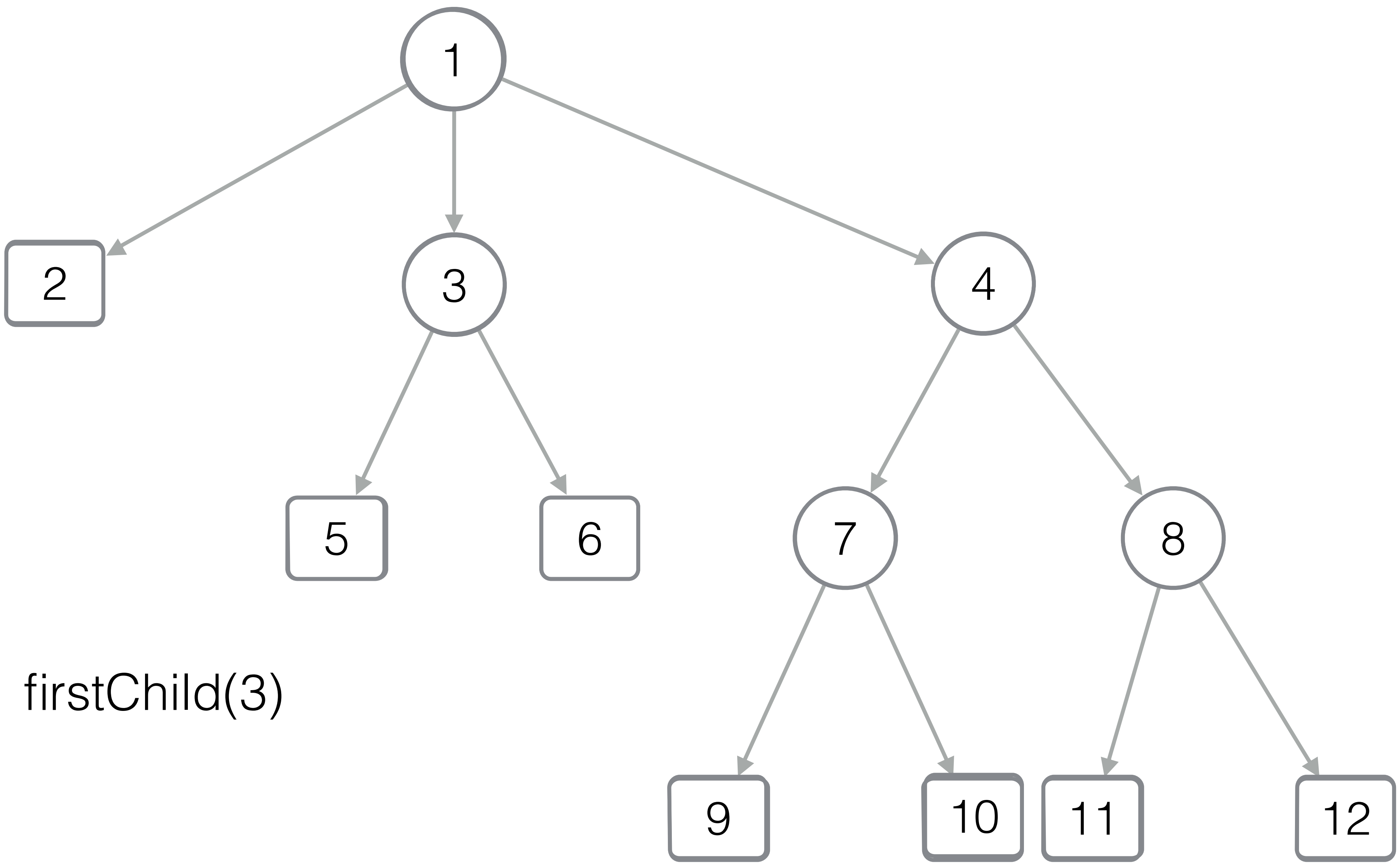
[LOUDS - Level-order unary degree sequence]

$pos(x) = Select_1(x)$

$firstChild(x) = ?$

$y = Select_0(x)+1$

// start of x's children in B



firstChild(3)

B 1 0 1 1 1 0 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 0 0

Succinct representation of trees (1)

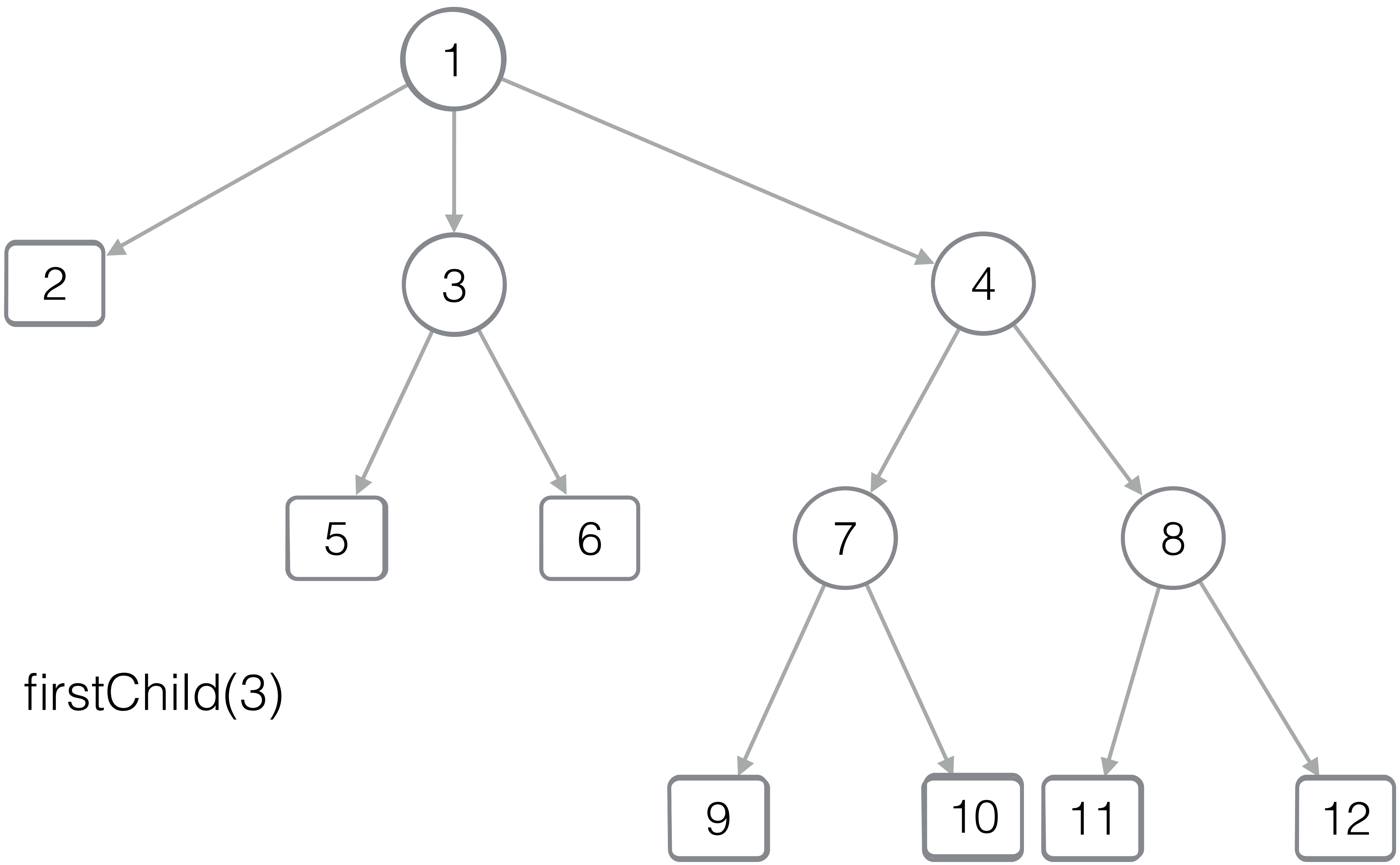
[LOUDS - Level-order unary degree sequence]

$pos(x) = Select_1(x)$

$firstChild(x) = ?$

$y = Select_0(x)+1$

// start of x's children in B



firstChild(3)

$y = Select_0(3)+1=8$

B 1 0 1 1 1 0 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 0 0

1 2 3 4 5 6 7 8 9 10 11 12

Succinct representation of trees (1)

[LOUDS - Level-order unary degree sequence]

$pos(x) = Select_1(x)$

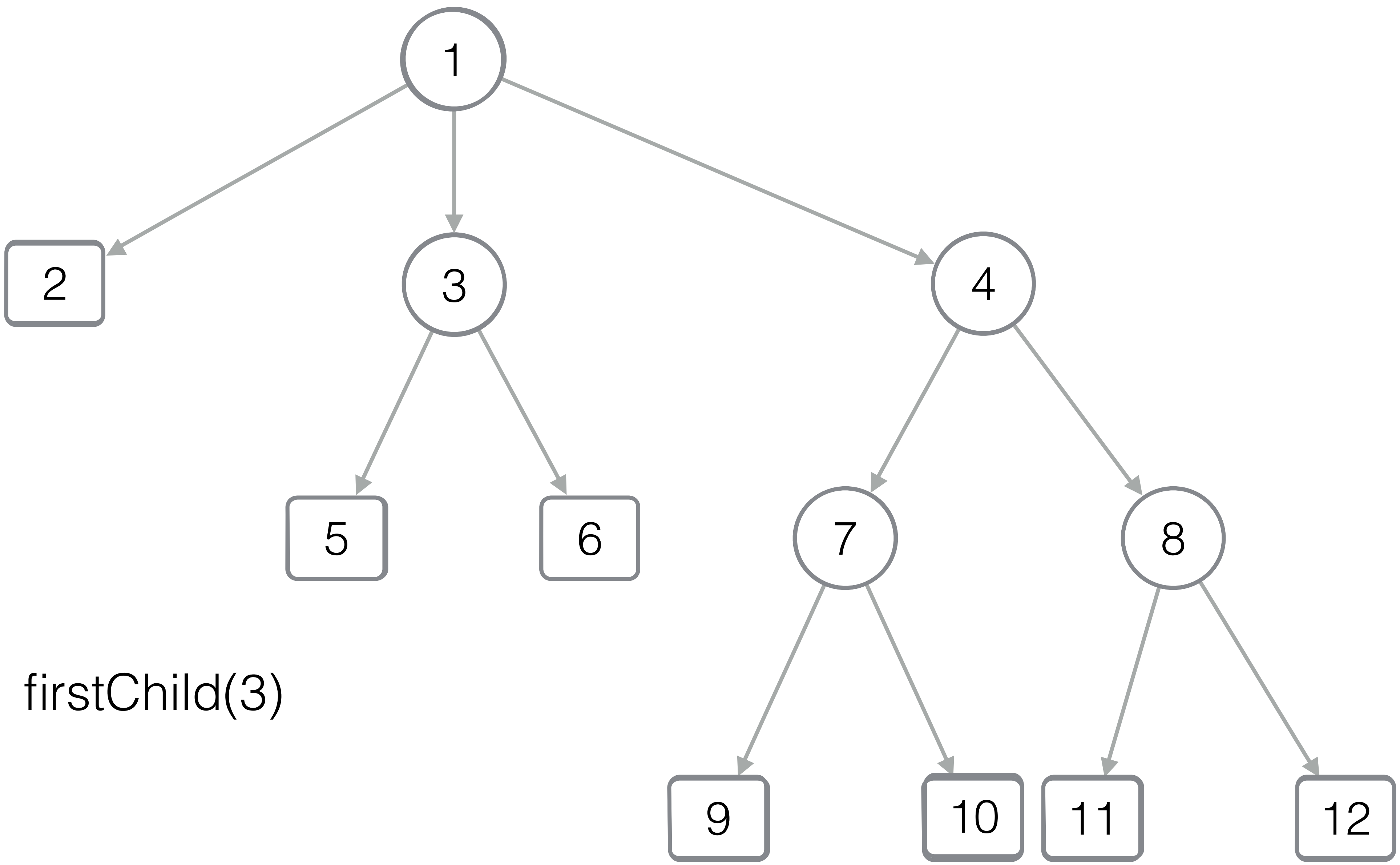
$firstChild(x) = ?$

$y = Select_0(x)+1$

// start of x's children in B

if $B[y] == 0$

return -1 // is a leaf



firstChild(3)

$y = Select_0(3)+1=8$

B 1 0 1 1 1 0 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 0 0

(1) (2) (3) (4) (5) (6) (7) (8) (9) (10) (11) (12)

Succinct representation of trees (1)

[LOUDS - Level-order unary degree sequence]

$pos(x) = Select_1(x)$

firstChild(x) = ?

$y = Select_0(x)+1$

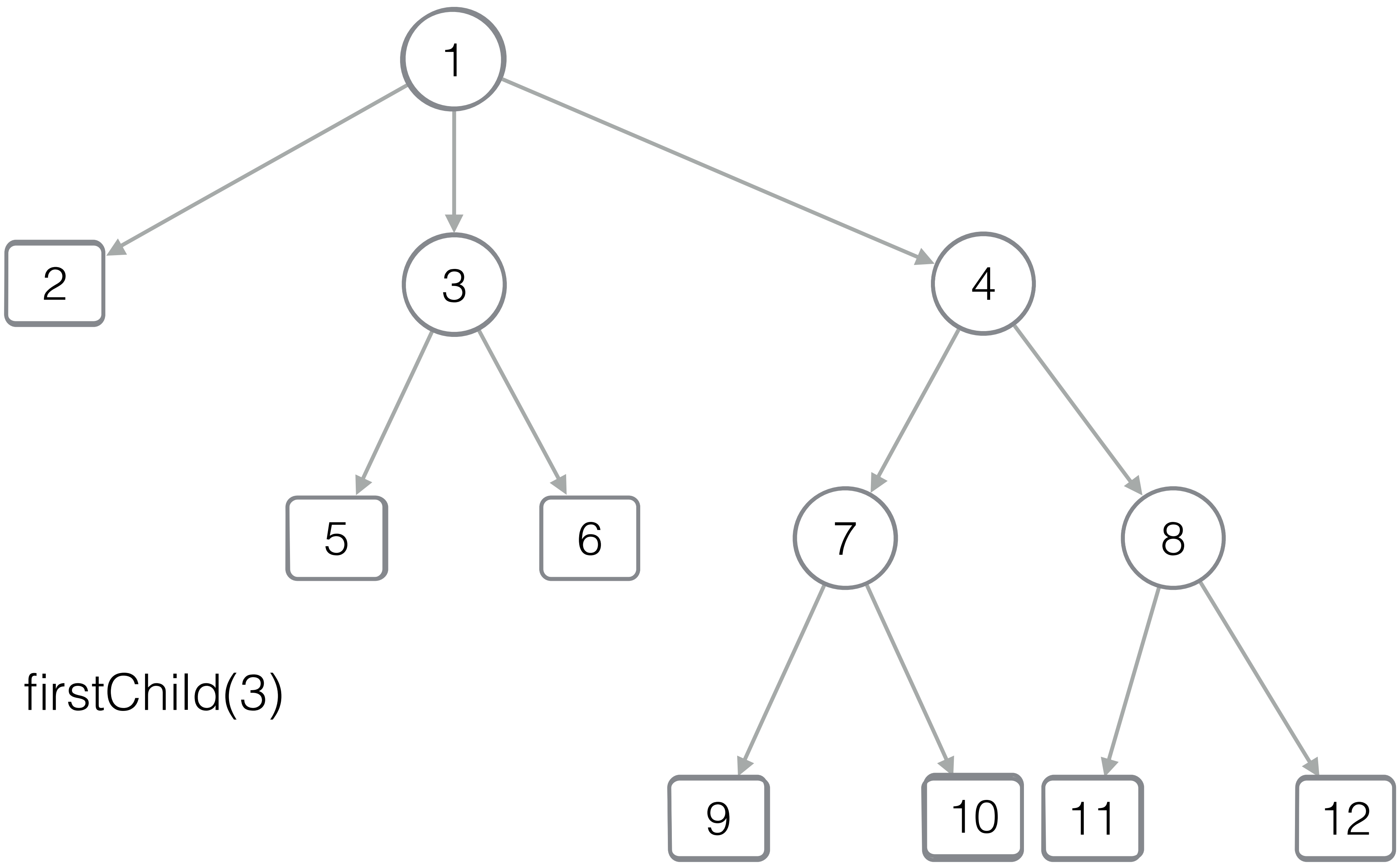
// start of x's children in B

if B[y] == 0

return -1 // is a leaf

else

return y-x // $Rank_1(y)$



firstChild(3)

$y = Select_0(3)+1=8$

B 1 0 1 1 1 0 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 0 0

1 2 3 4 5 6 7 8 9 10 11 12

Succinct representation of trees (1)

[LOUDS - Level-order unary degree sequence]

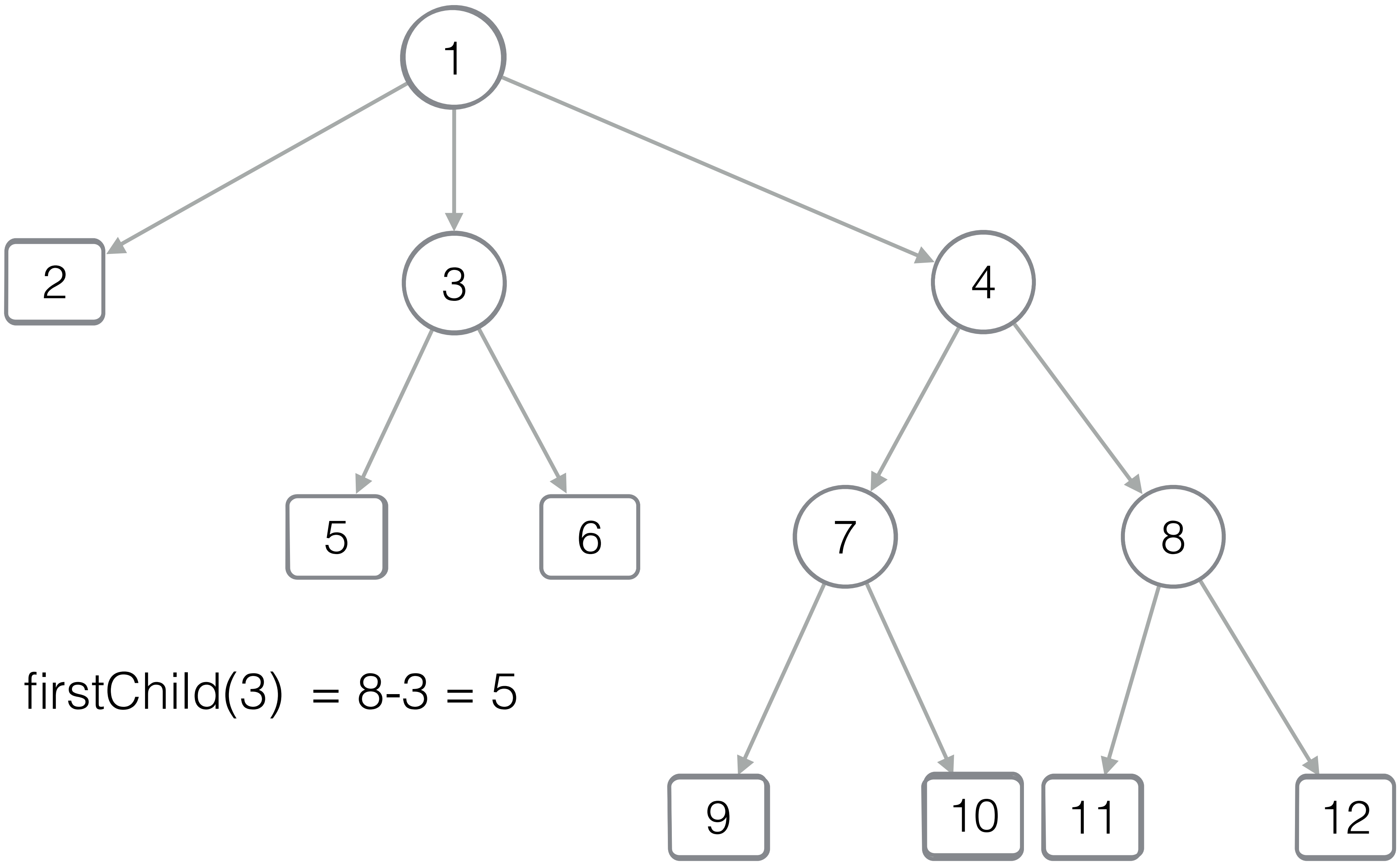
$pos(x) = Select_1(x)$

firstChild(x) = ?

$y = Select_0(x)+1$

// start of x's children in B

```
if B[y] == 0
    return -1   // is a leaf
else
    return y-x  // Rank1(y)
```



firstChild(3) = 8-3 = 5

$y = Select_0(3)+1=8$

B 1 0 1 1 1 0 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 0 0

(1) (2) (3) (4) (5) (6) (7) (8) (9) (10) (11) (12)

Succinct representation of trees (1)

[LOUDS - Level-order unary degree sequence]

$pos(x) = Select_1(x)$

firstChild(x) = ?

$y = Select_0(x)+1$

// start of x's children in B

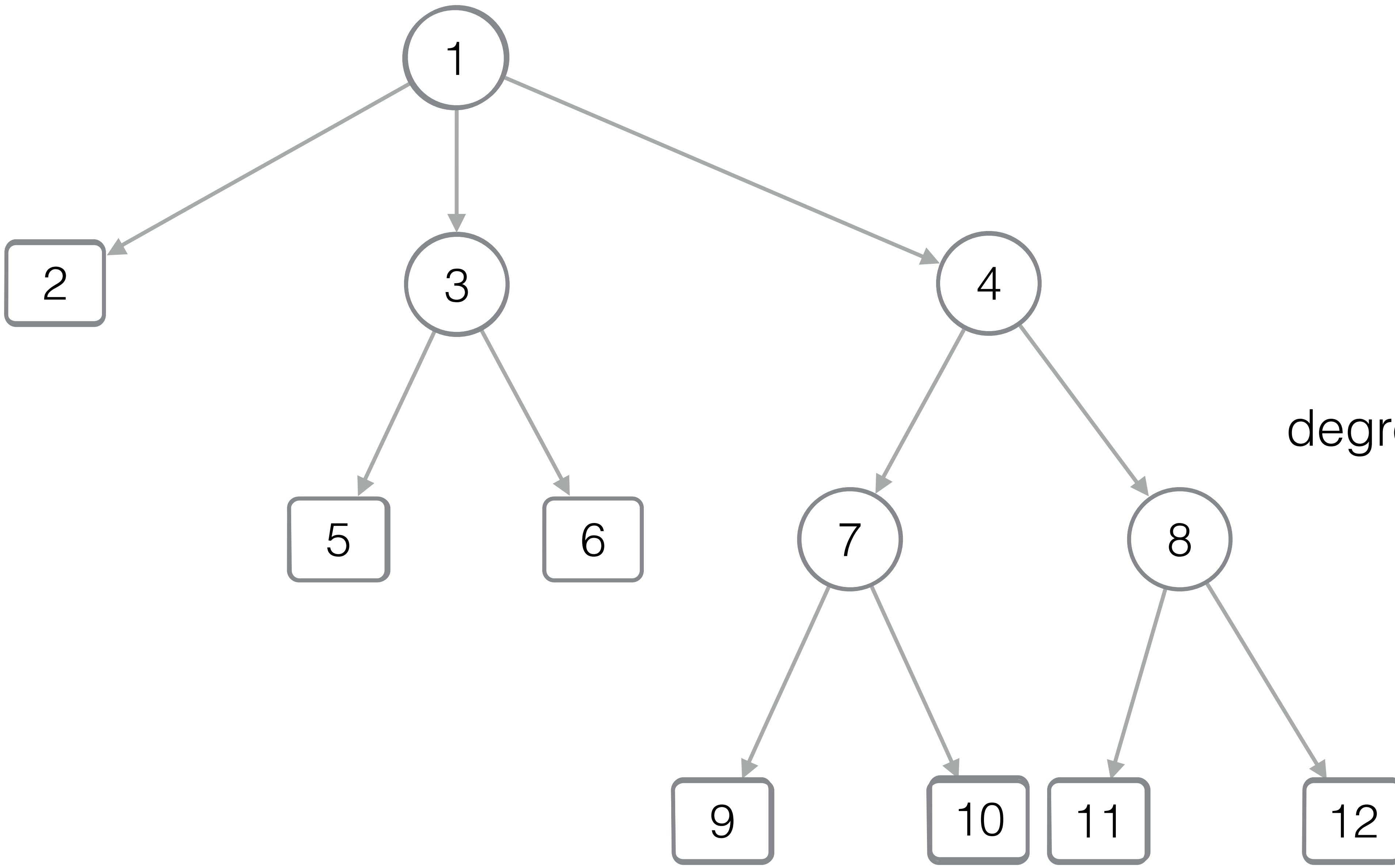
if B[y] == 0

return -1 // is a leaf

else

return y-x // $Rank_1(y)$

degree(x) = ?



$y= Select_0(3)+1=8$

B 1 0 1 1 1 0 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 0 0

1 2 3 4 5 6 7 8 9 10 11 12

Succinct representation of trees (1)

[LOUDS - Level-order unary degree sequence]

$pos(x) = Select_1(x)$

$firstChild(x) = ?$

$y = Select_0(x)+1$

// start of x's children in B

if $B[y] == 0$

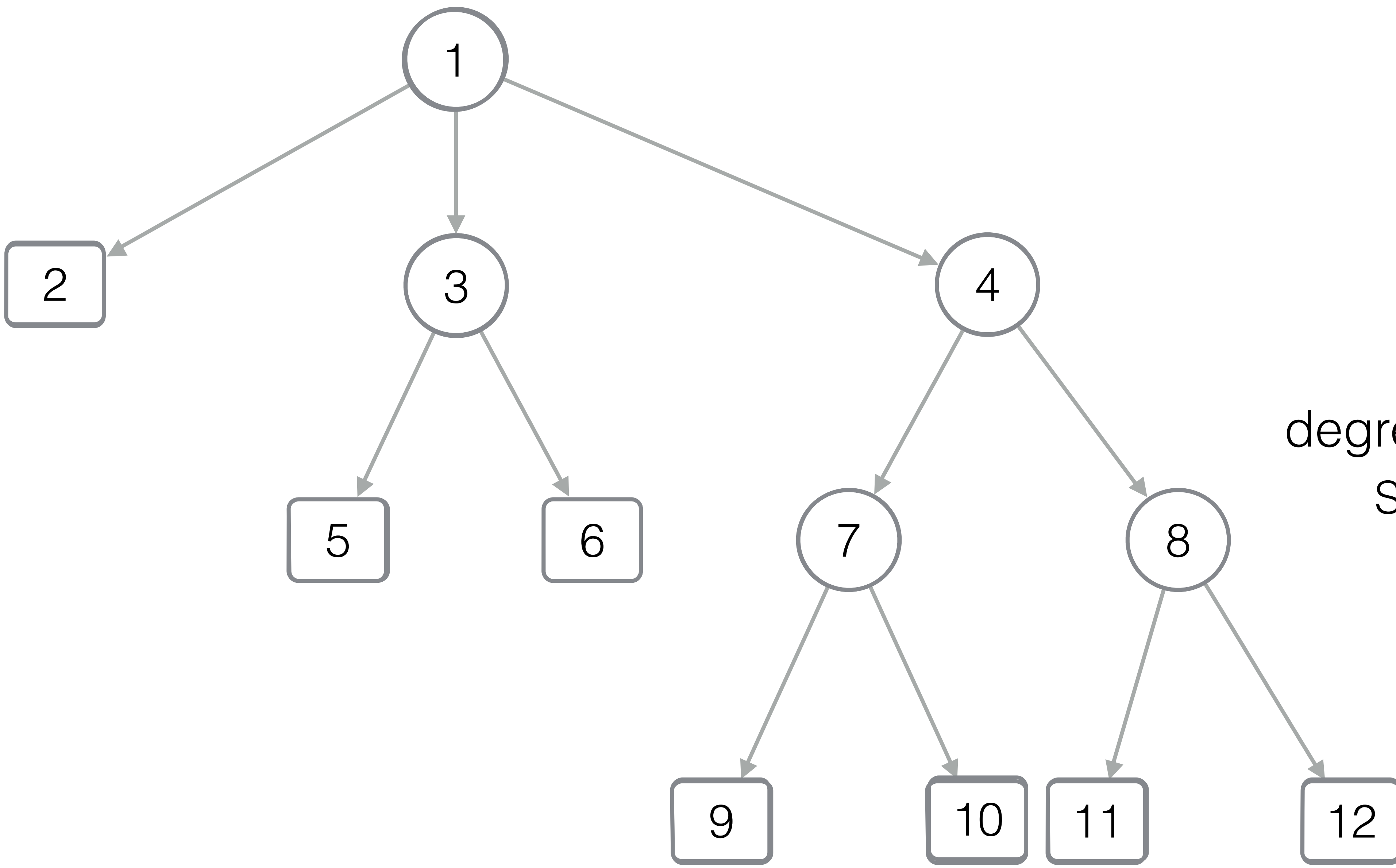
return -1 // is a leaf

else

return $y-x$ // $Rank_1(y)$

$degree(x) = ?$

$Select_0(x+1) - (Select_0(x) + 1)$



$y= Select_0(3)+1=8$

B 1 0 1 1 1 0 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 0 0

(1) (2) (3) (4) (5) (6) (7) (8) (9) (10) (11) (12)

Succinct representation of trees (1)

[LOUDS - Level-order unary degree sequence]

$pos(x) = Select_1(x)$

$firstChild(x) = ?$

$y = Select_0(x)+1$

// start of x's children in B

if $B[y] == 0$

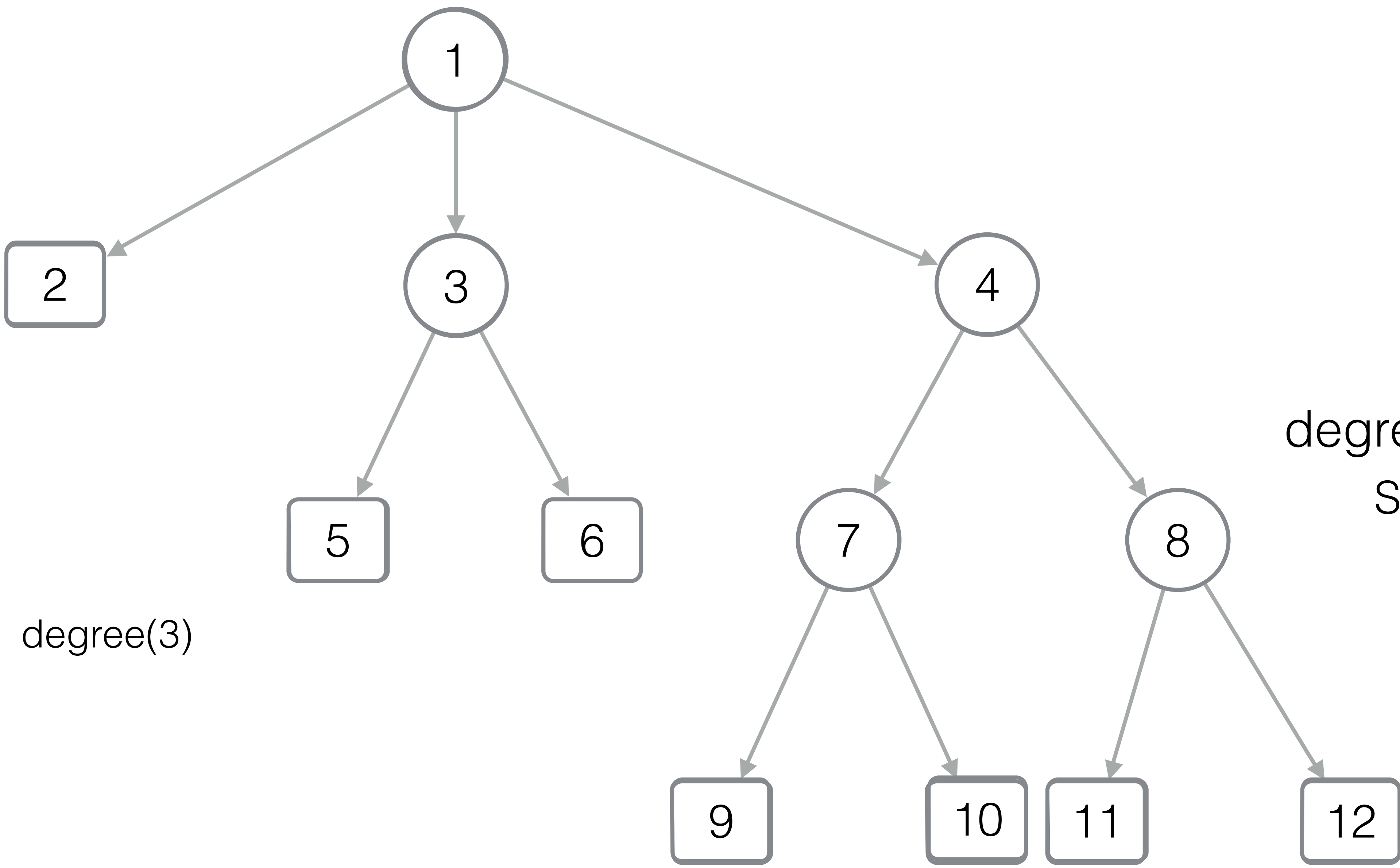
return -1 // is a leaf

else

return y-x // $Rank_1(y)$

$degree(x) = ?$

$Select_0(x+1) - (Select_0(x) + 1)$



degree(3)

$y = Select_0(3)+1=8$

B 1 0 1 1 1 0 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 0 0

1 2 3 4 5 6 7 8 9 10 11 12

Succinct representation of trees (1)

[LOUDS - Level-order unary degree sequence]

$pos(x) = Select_1(x)$

$firstChild(x) = ?$

$y = Select_0(x)+1$

// start of x's children in B

if $B[y] == 0$

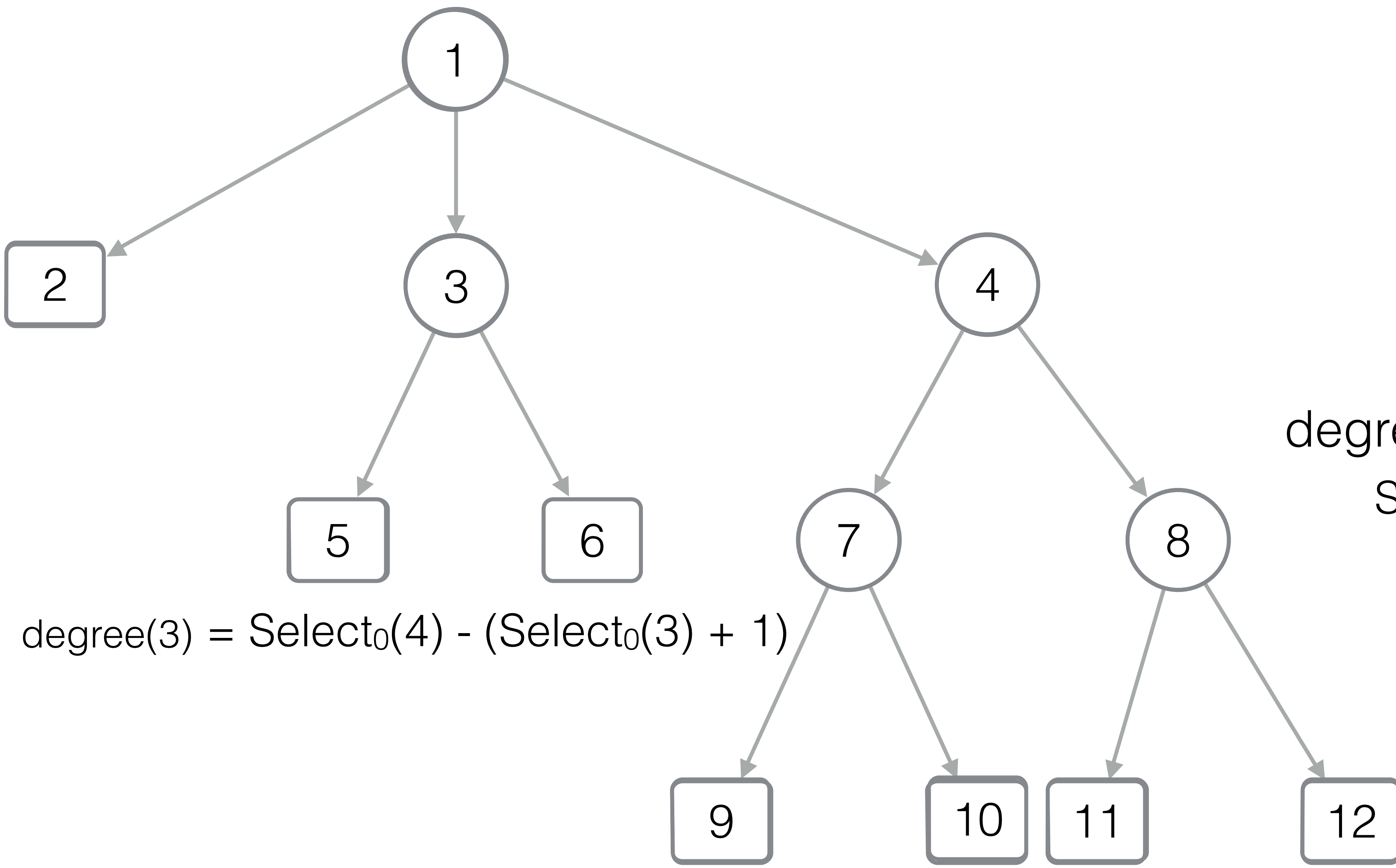
return -1 // is a leaf

else

return y-x // $Rank_1(y)$

$degree(x) = ?$

$Select_0(x+1) - (Select_0(x) + 1)$



$degree(3) = Select_0(4) - (Select_0(3) + 1)$

$y = Select_0(3)+1=8$

B 1 0 1 1 1 0 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 0 0

(1) (2) (3) (4) (5) (6) (7) (8) (9) (10) (11) (12)

Succinct representation of trees (1)

[LOUDS - Level-order unary degree sequence]

$pos(x) = Select_1(x)$

$firstChild(x) = ?$

$y = Select_0(x)+1$

// start of x's children in B

if $B[y] == 0$

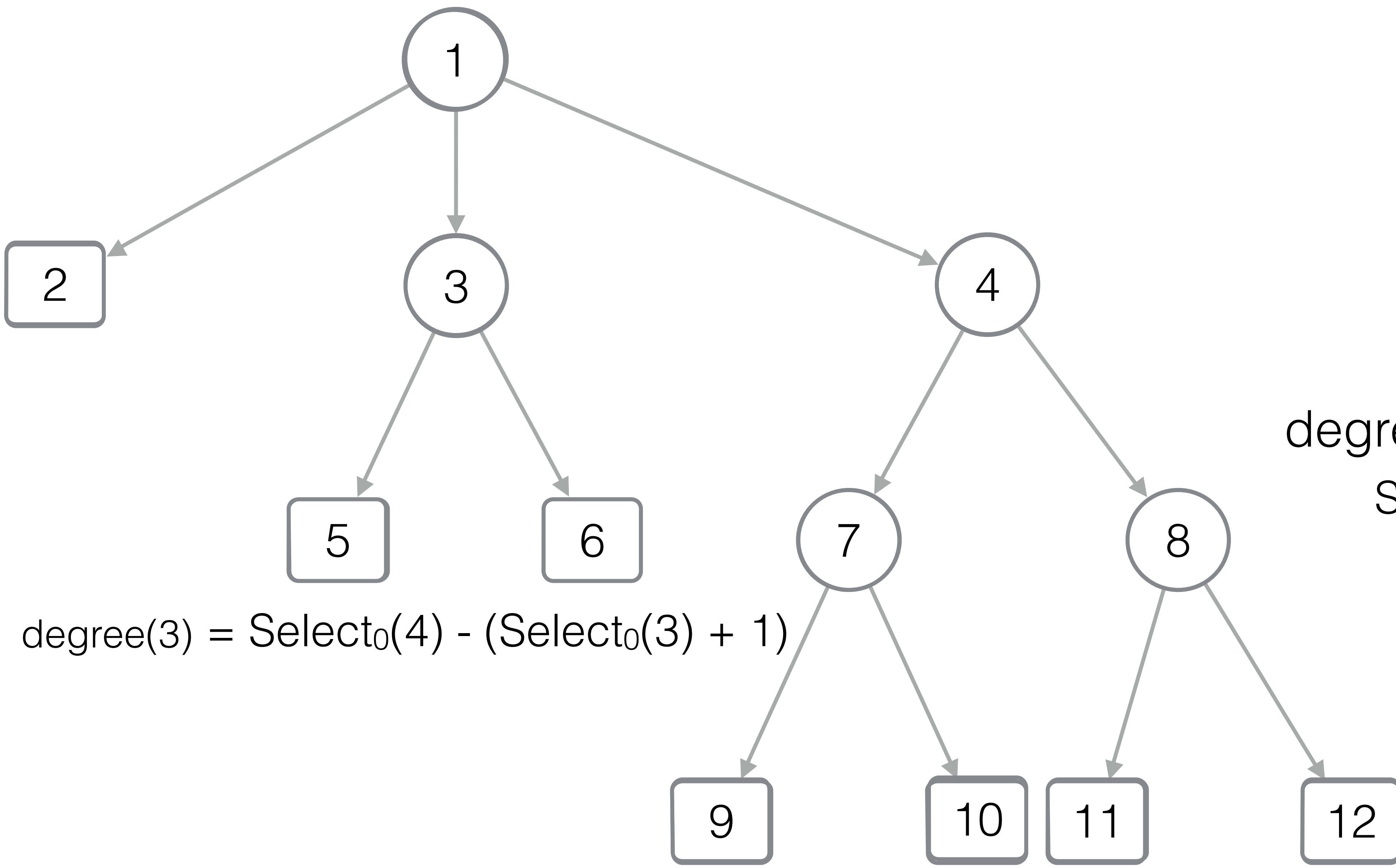
return -1 // is a leaf

else

return y-x // $Rank_1(y)$

$degree(x) = ?$

$Select_0(x+1) - (Select_0(x) + 1)$



$degree(3) = Select_0(4) - (Select_0(3) + 1)$

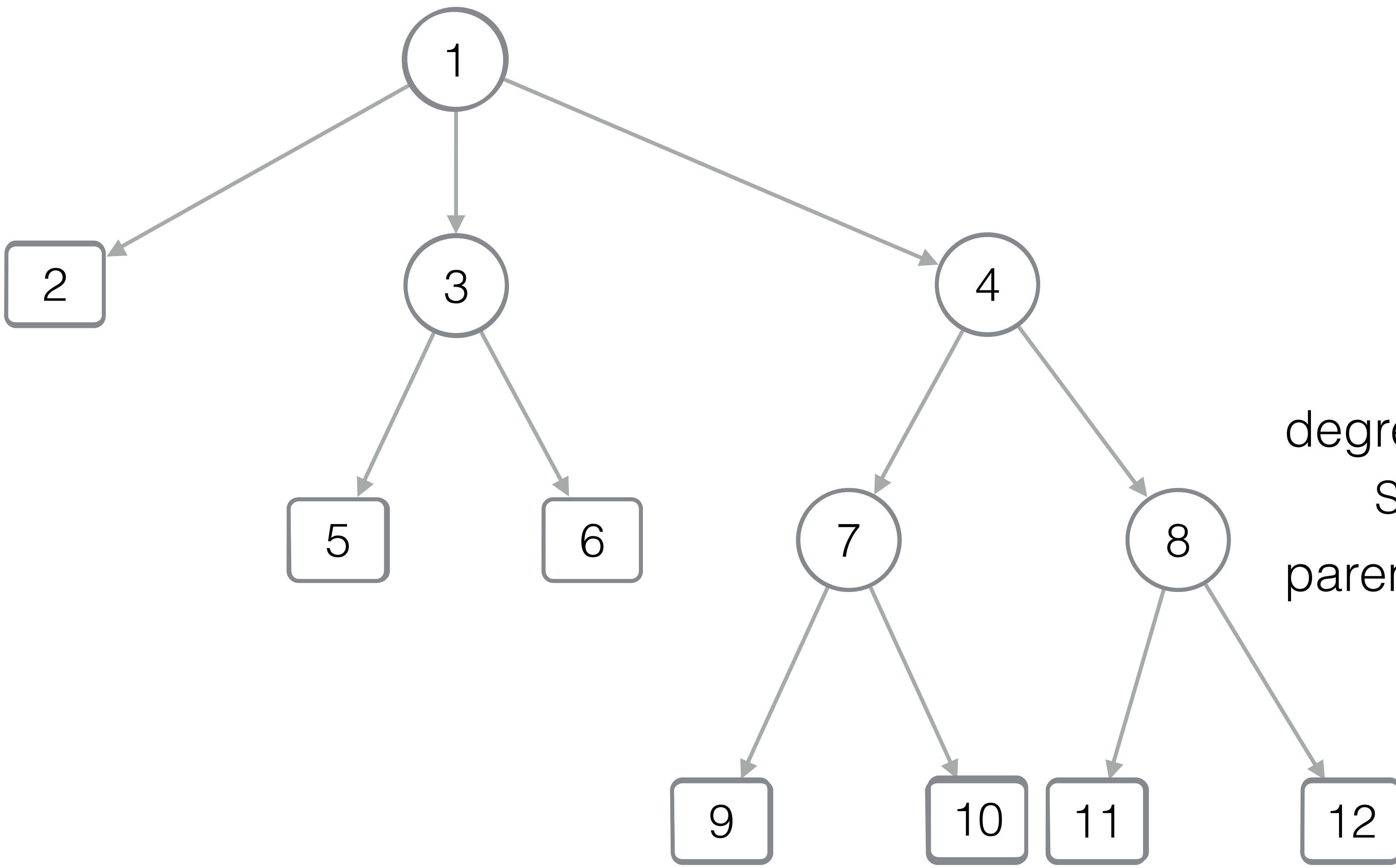
$y= Select_0(3)+1=8$ $Select_0(4)=10$

B 1 0 1 1 1 0 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 0 0

1 2 3 4 5 6 7 8 9 10 11 12

Succinct representation of trees (1)

[LOUDS - Level-order unary degree sequence]



$pos(x) = Select_1(x)$

$firstChild(x) = ?$

$y = Select_0(x)+1$

// start of x's children in B

if B[y] == 0

return -1 // is a leaf

else

return y-x // $Rank_1(y)$

$degree(x) = ?$

$Select_0(x+1) - (Select_0(x) + 1)$

parent(x) =

B 1 0 1 1 1 0 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 0 0

1 2 3 4 5 6 7 8 9 10 11 12

Succinct representation of trees (1)

[LOUDS - Level-order unary degree sequence]

$pos(x) = Select_1(x)$

$firstChild(x) = ?$

$y = Select_0(x)+1$

// start of x's children in B

if $B[y] == 0$

return -1 // is a leaf

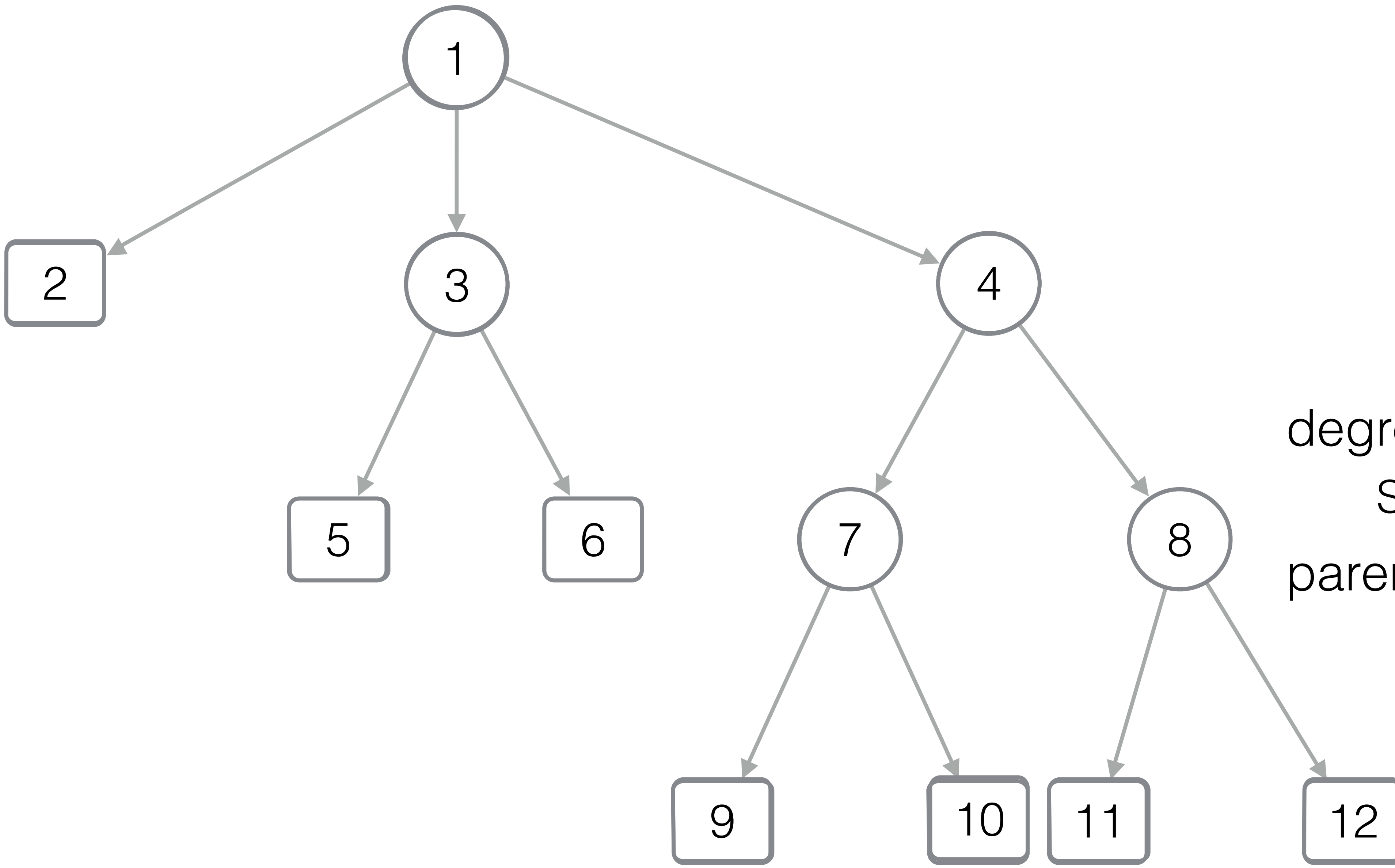
else

return $y-x$ // $Rank_1(y)$

$degree(x) = ?$

$Select_0(x+1) - (Select_0(x) + 1)$

$parent(x) = Rank_0(pos(x))$



B 1 0 1 1 1 0 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 0 0

1 2 3 4 5 6 7 8 9 10 11 12

Succinct representation of trees (1)

[LOUDS - Level-order unary degree sequence]

$pos(x) = Select_1(x)$

$firstChild(x) = ?$

$y = Select_0(x)+1$

// start of x's children in B

if $B[y] == 0$

return -1 // is a leaf

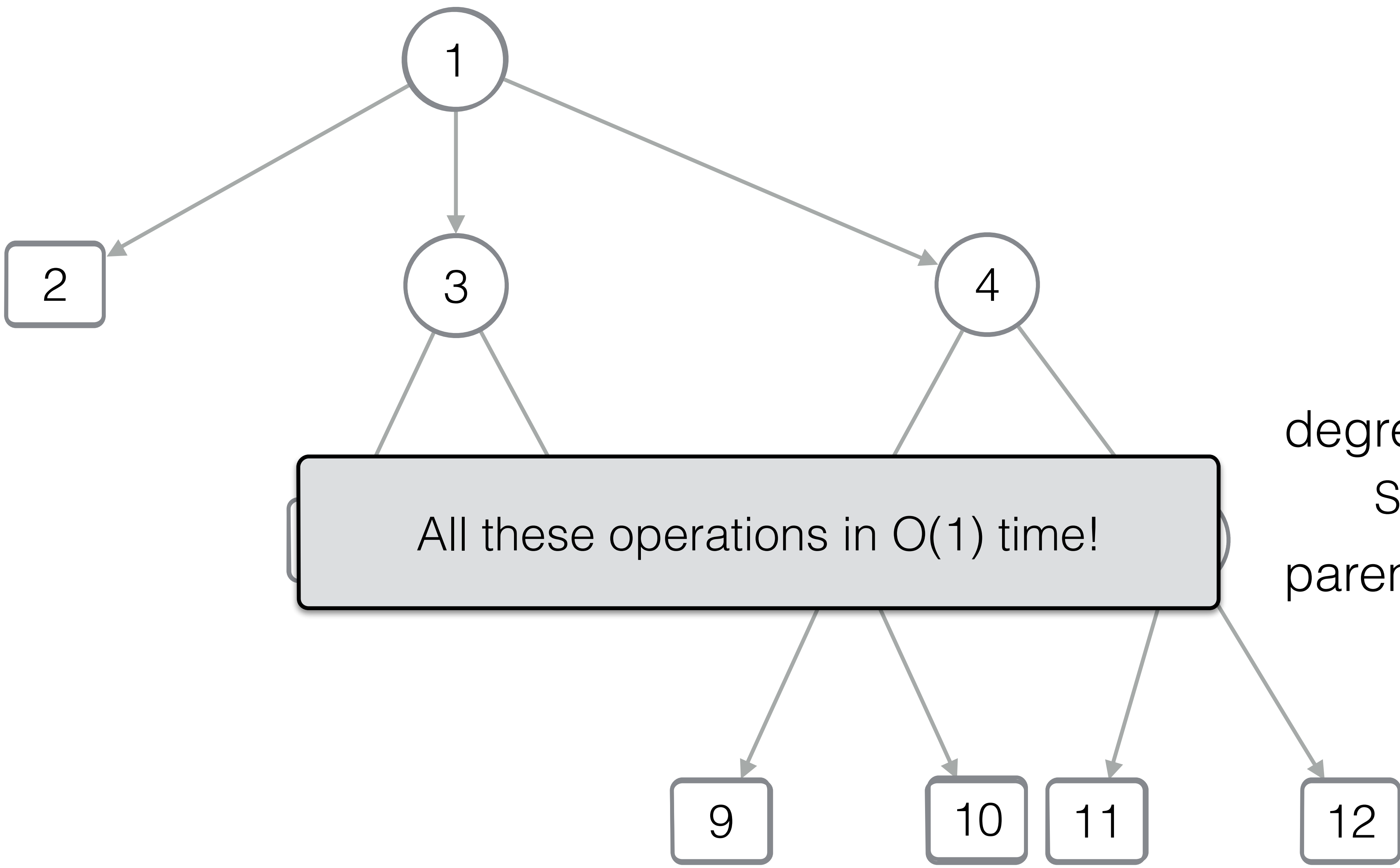
else

return y-x // $Rank_1(y)$

$degree(x) = ?$

$Select_0(x+1) - (Select_0(x) + 1)$

$parent(x) = Rank_0(pos(x))$

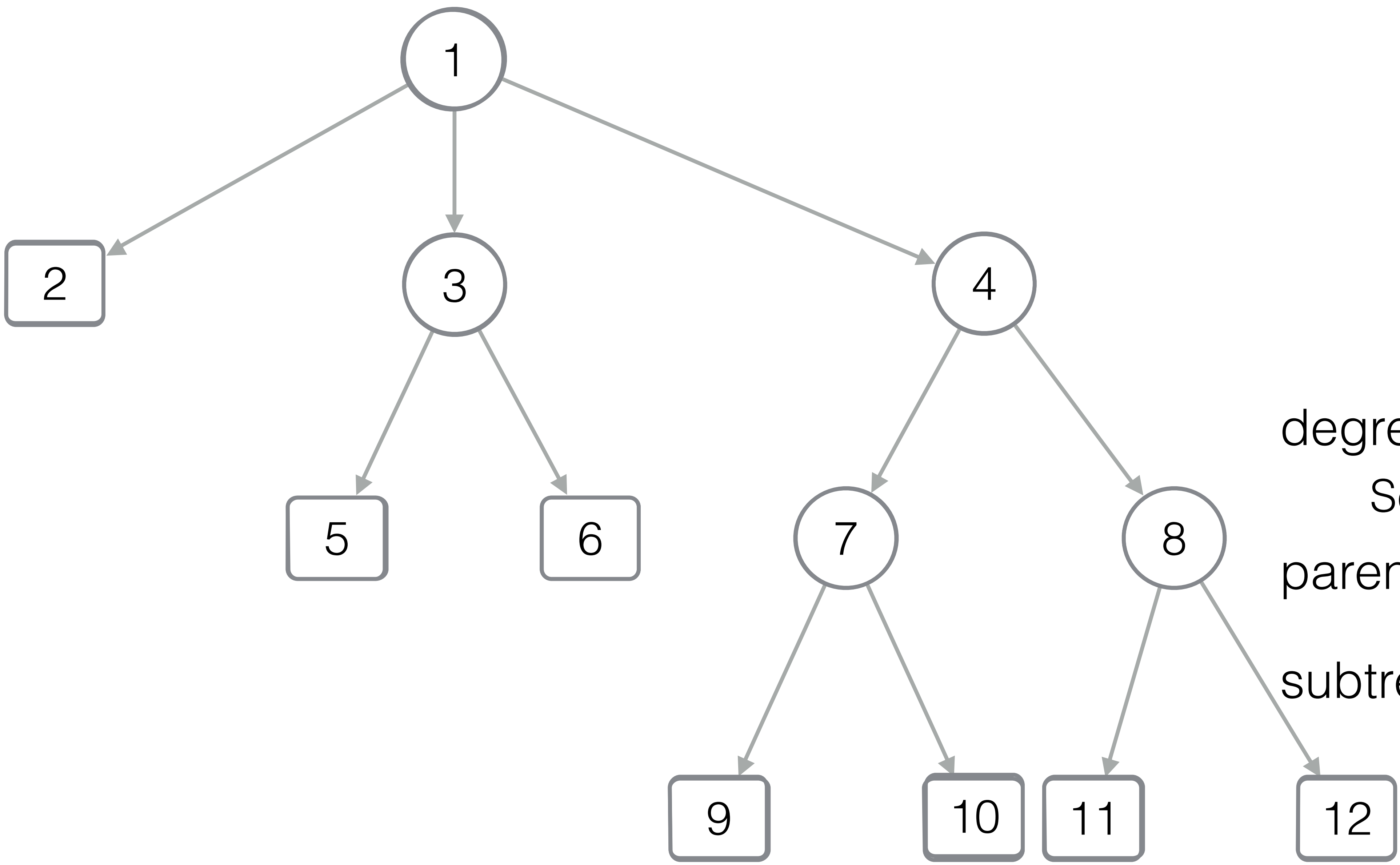


All these operations in O(1) time!

B 1 0 1 1 1 0 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 0 0

1 2 3 4 5 6 7 8 9 10 11 12

Succinct representation of trees (1)



[LOUDS - Level-order unary degree sequence]

$pos(x) = Select_1(x)$

$firstChild(x) = ?$

$y = Select_0(x)+1$

// start of x's children in B

if B[y] == 0

return -1 // is a leaf

else

return y-x // $Rank_1(y)$

$degree(x) = ?$

$Select_0(x+1) - (Select_0(x) + 1)$

$parent(x) = Rank_0(pos(x))$

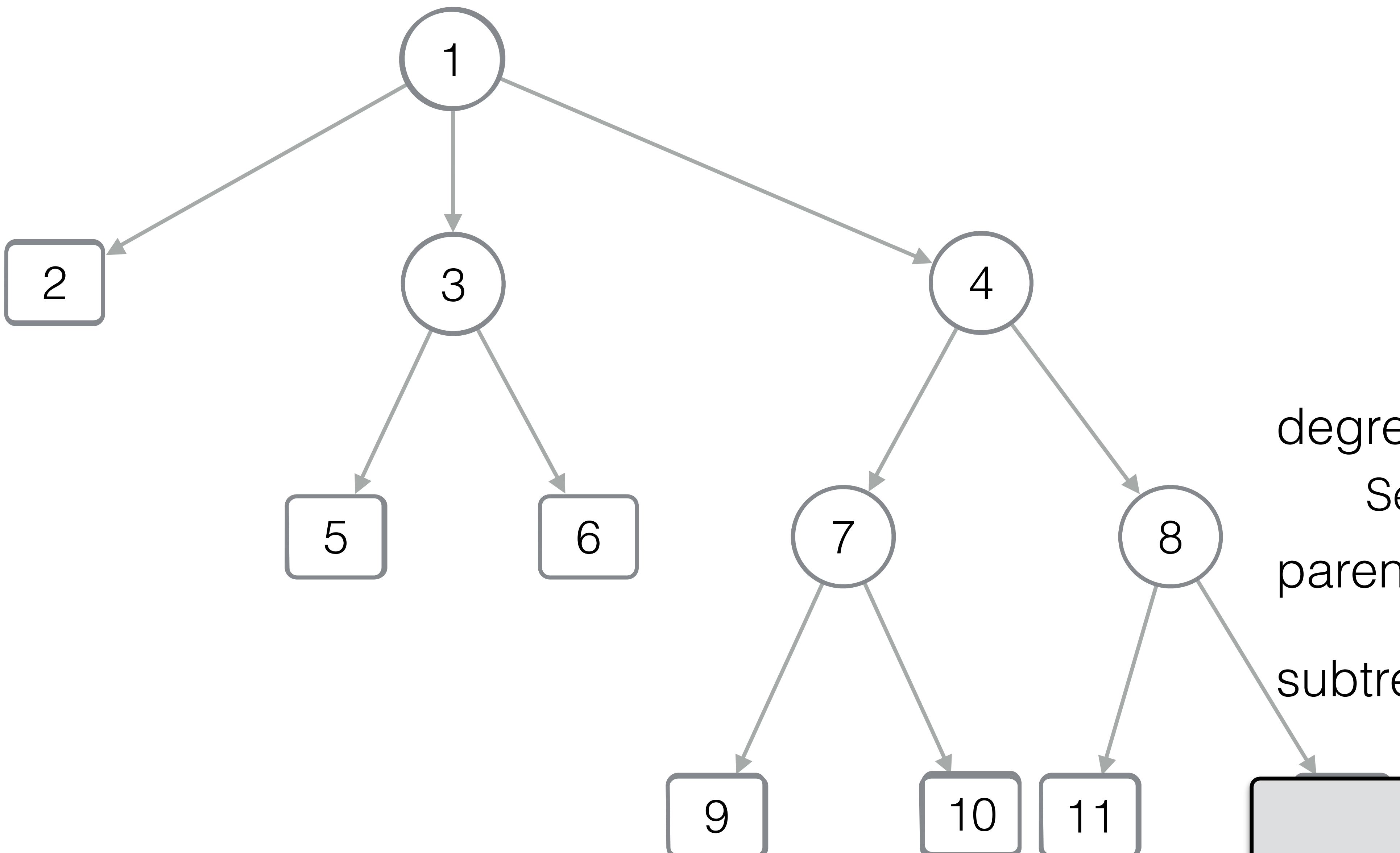
$subtreeSize(x) = ?$

B 1 0 1 1 1 0 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 0 0

1 2 3 4 5 6 7 8 9 10 11 12

Succinct representation of trees (1)

[LOUDS - Level-order unary degree sequence]



$pos(x) = Select_1(x)$

$firstChild(x) = ?$

$y = Select_0(x)+1$

// start of x's children in B

if $B[y] == 0$

return -1 // is a leaf

else

return y-x // $Rank_1(y)$

$degree(x) = ?$

$Select_0(x+1) - (Select_0(x) + 1)$

$parent(x) = Rank_0(pos(x))$

$subtreeSize(x) = ?$

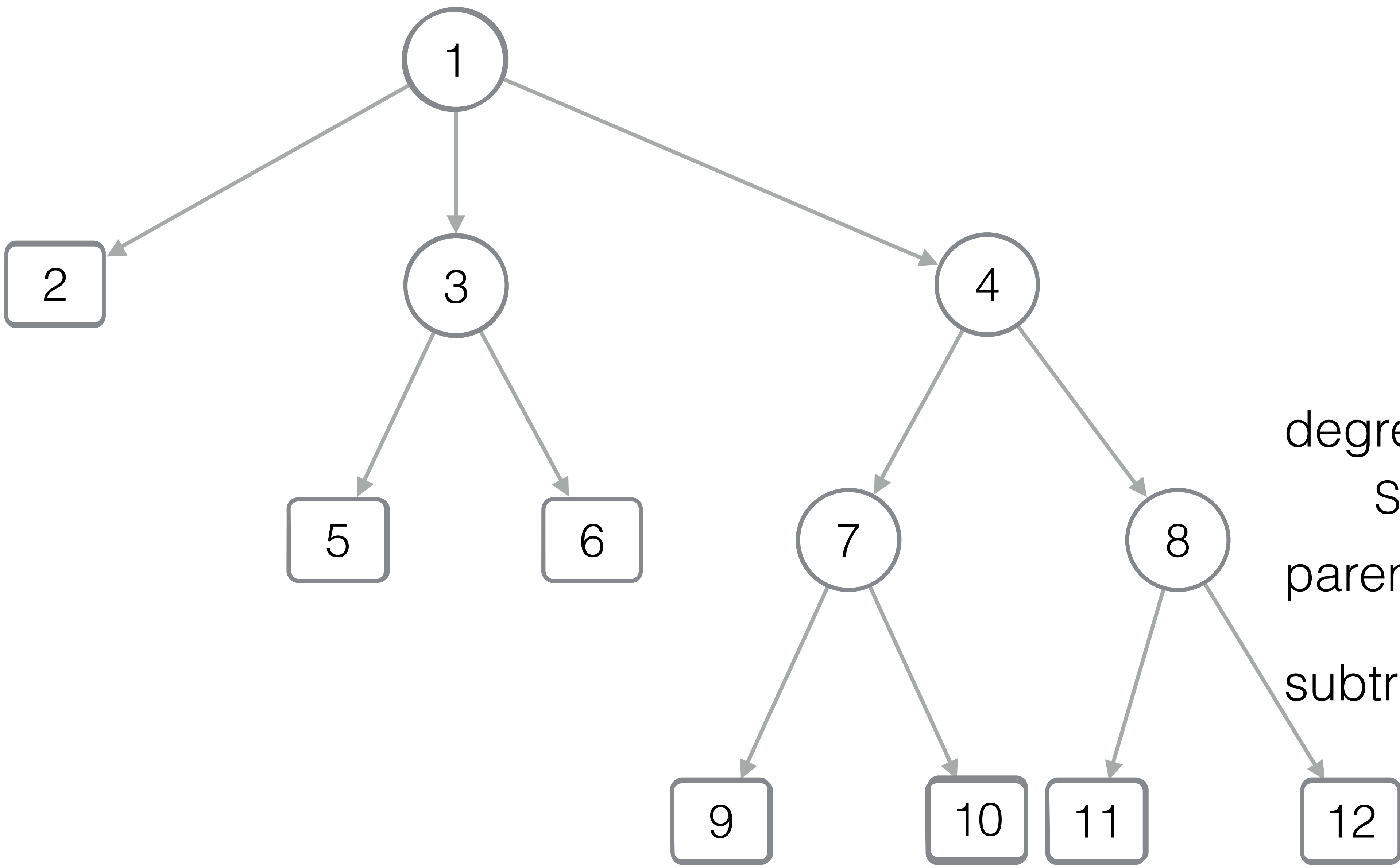
Not efficient!
Nodes of the subtree are spread in B

B 1 0 1 1 1 0 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 0 0

(1) (2)(3)(4) (5)(6) (7)(8) (9)(10) (11)(12)

Succinct representation of trees (1)

[LOUDS - Level-order unary degree sequence]



$pos(x) = Select_1(x)$

$firstChild(x) = ?$

$y = Select_0(x)+1$

// start of x's children in B

if $B[y] == 0$

return -1 // is a leaf

else

return $y-x$ // $Rank_1(y)$

$degree(x) = ?$

$Select_0(x+1) - (Select_0(x) + 1)$

$parent(x) = Rank_0(pos(x))$

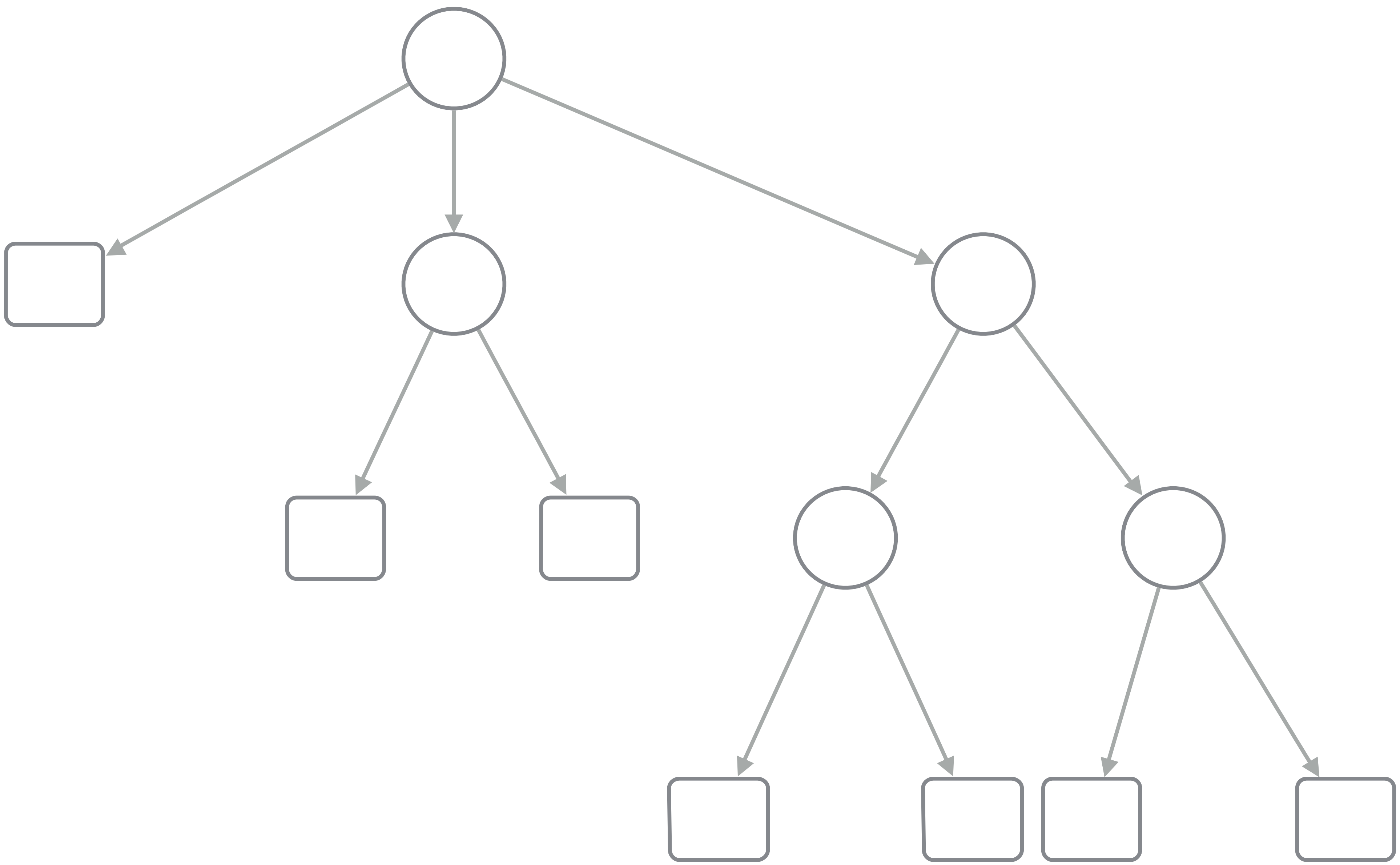
$subtreeSize(x) = ?$

B 1 0 1 1 1 0 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 0 0

1 2 3 4 5 6 7 8 9 10 11 12

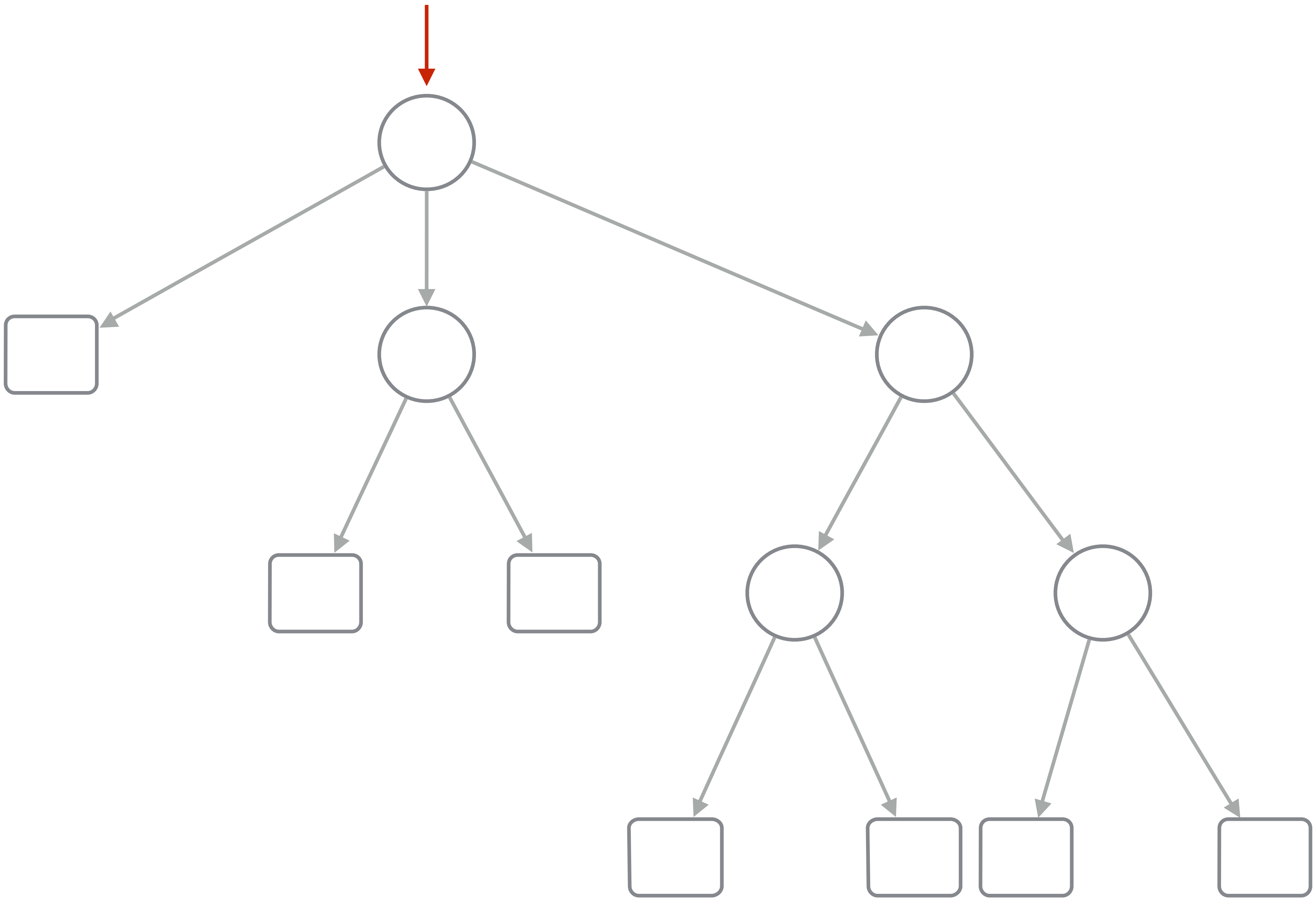
Succinct representation of trees (2)

[BP - Balanced parenthesis]



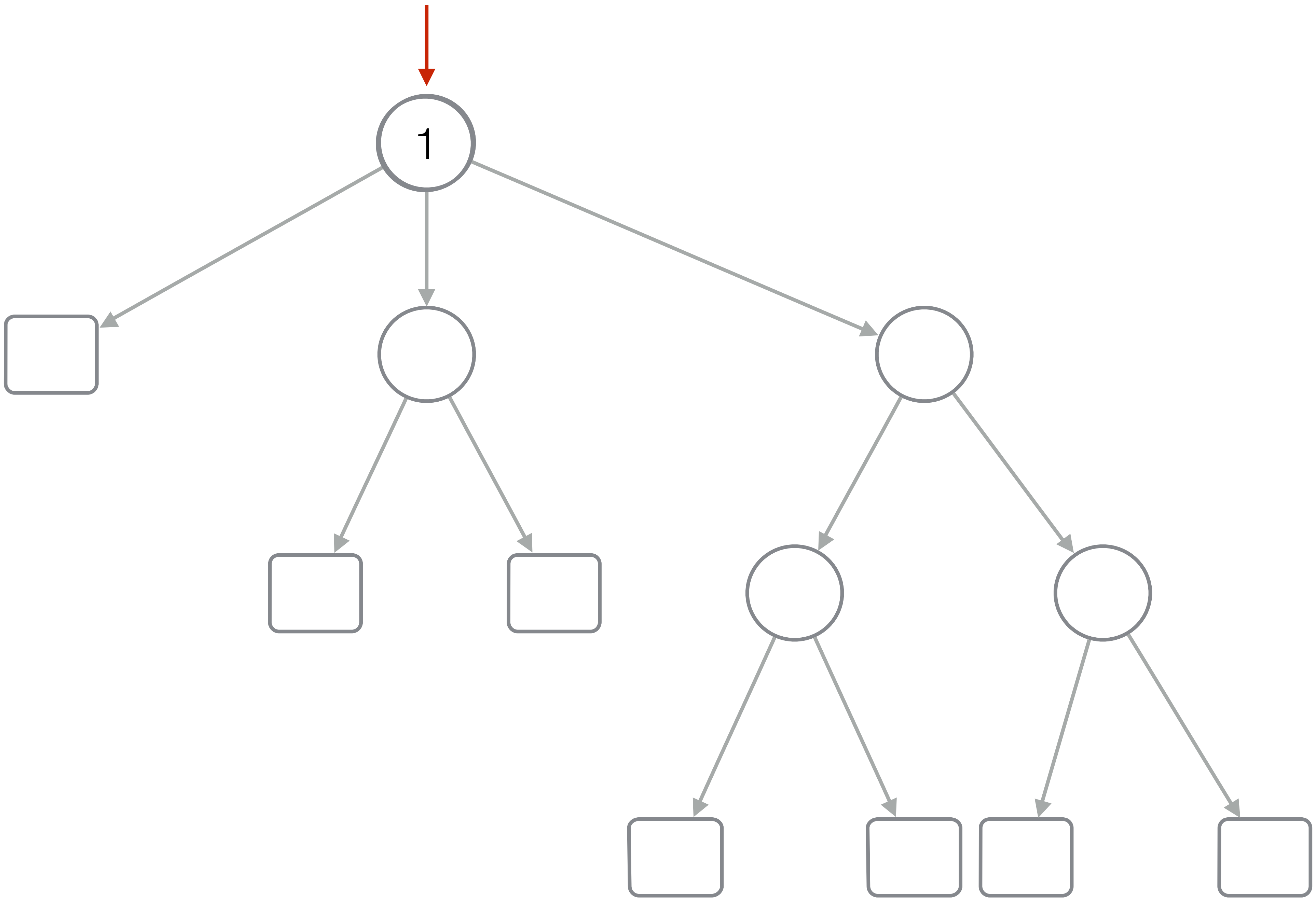
Succinct representation of trees (2)

[BP - Balanced parenthesis]



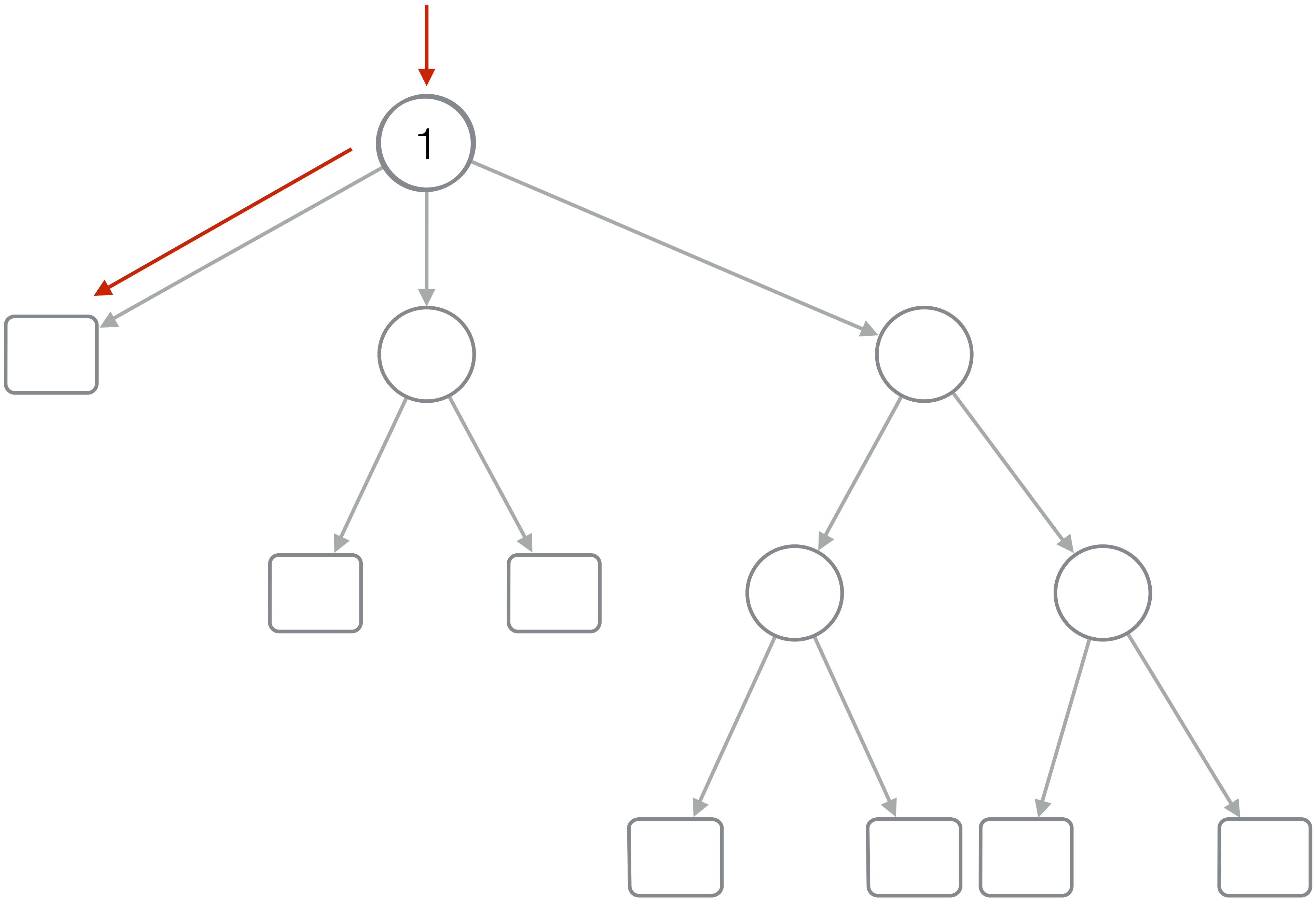
Succinct representation of trees (2)

[BP - Balanced parenthesis]



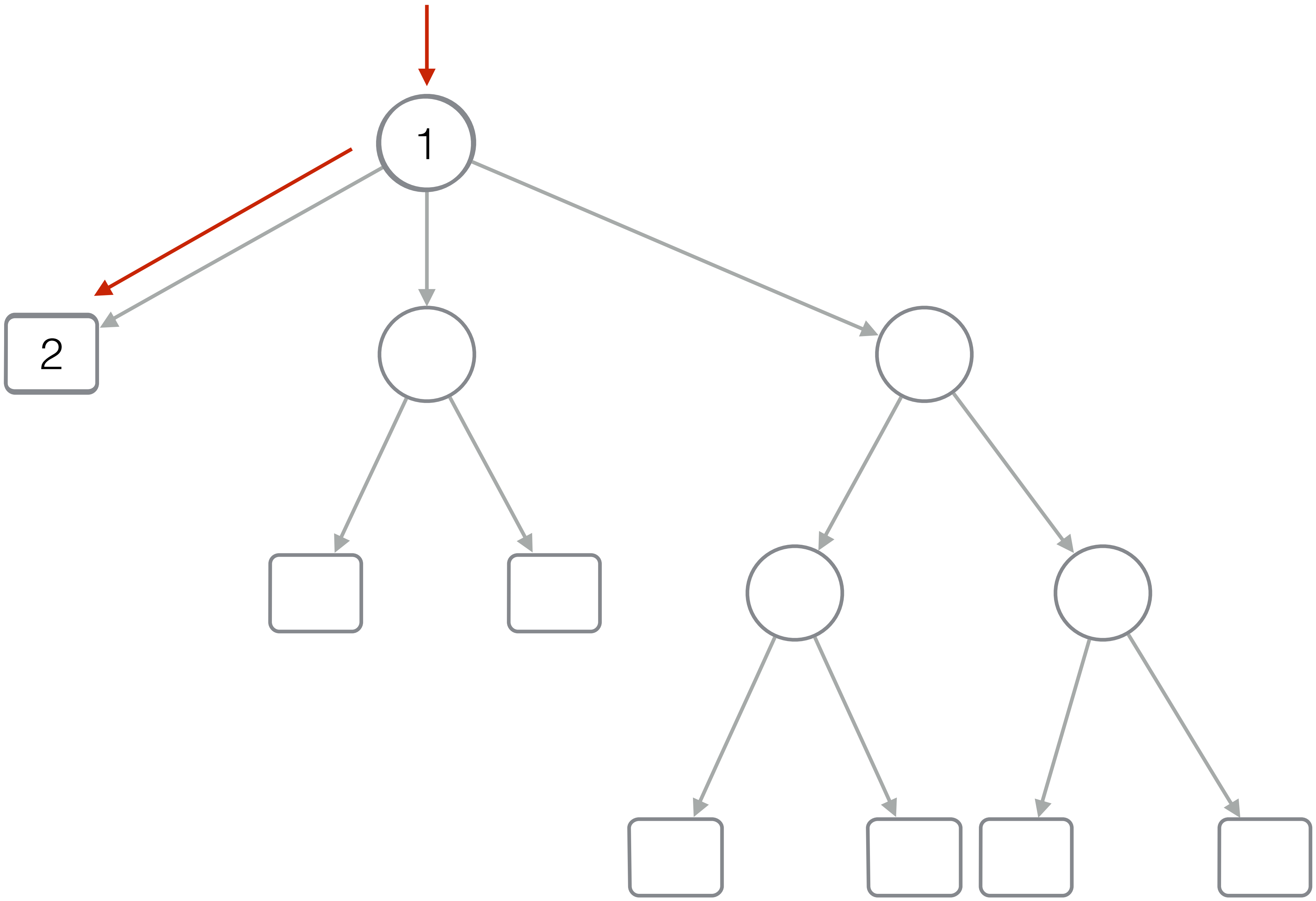
Succinct representation of trees (2)

[BP - Balanced parenthesis]



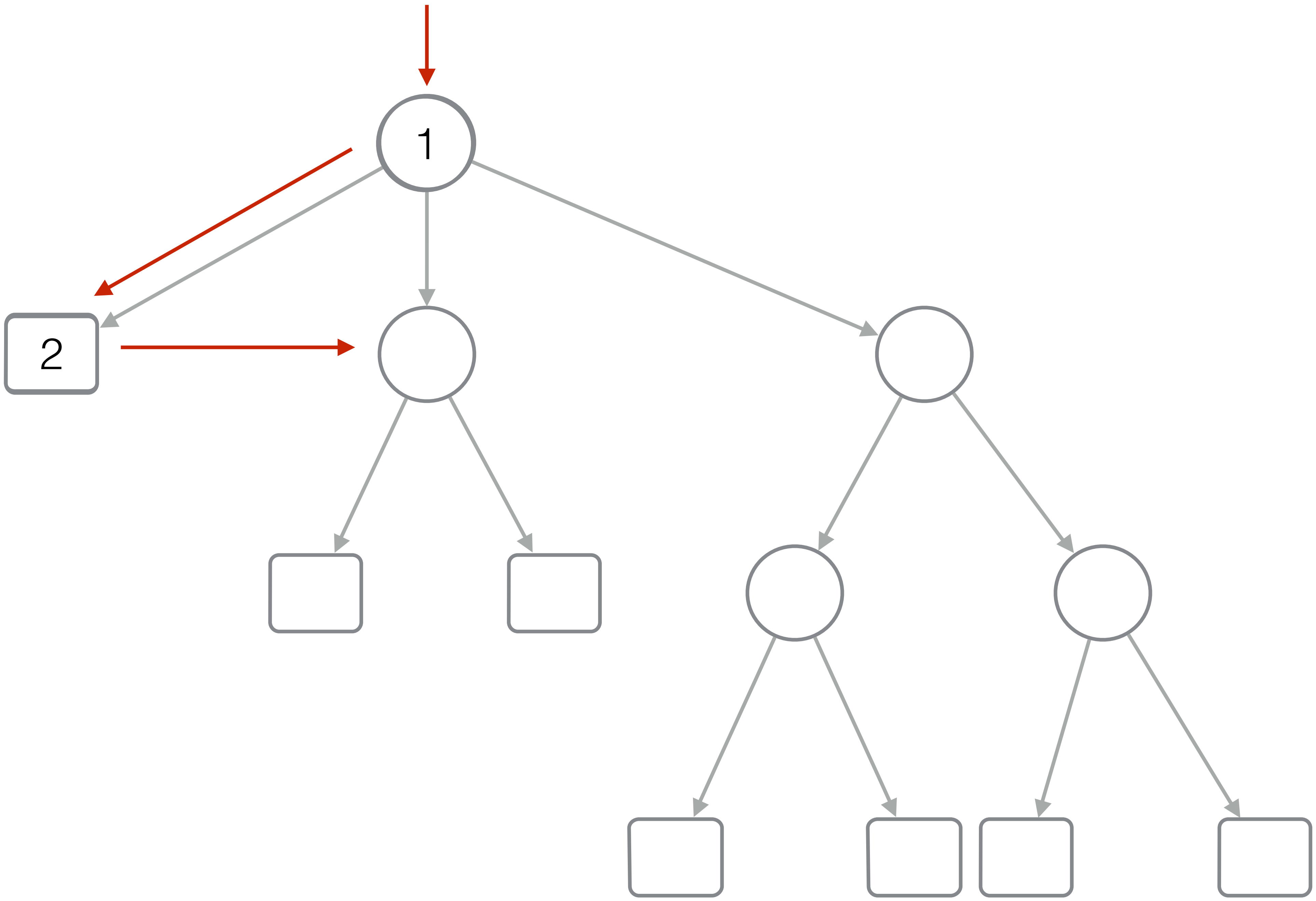
Succinct representation of trees (2)

[BP - Balanced parenthesis]



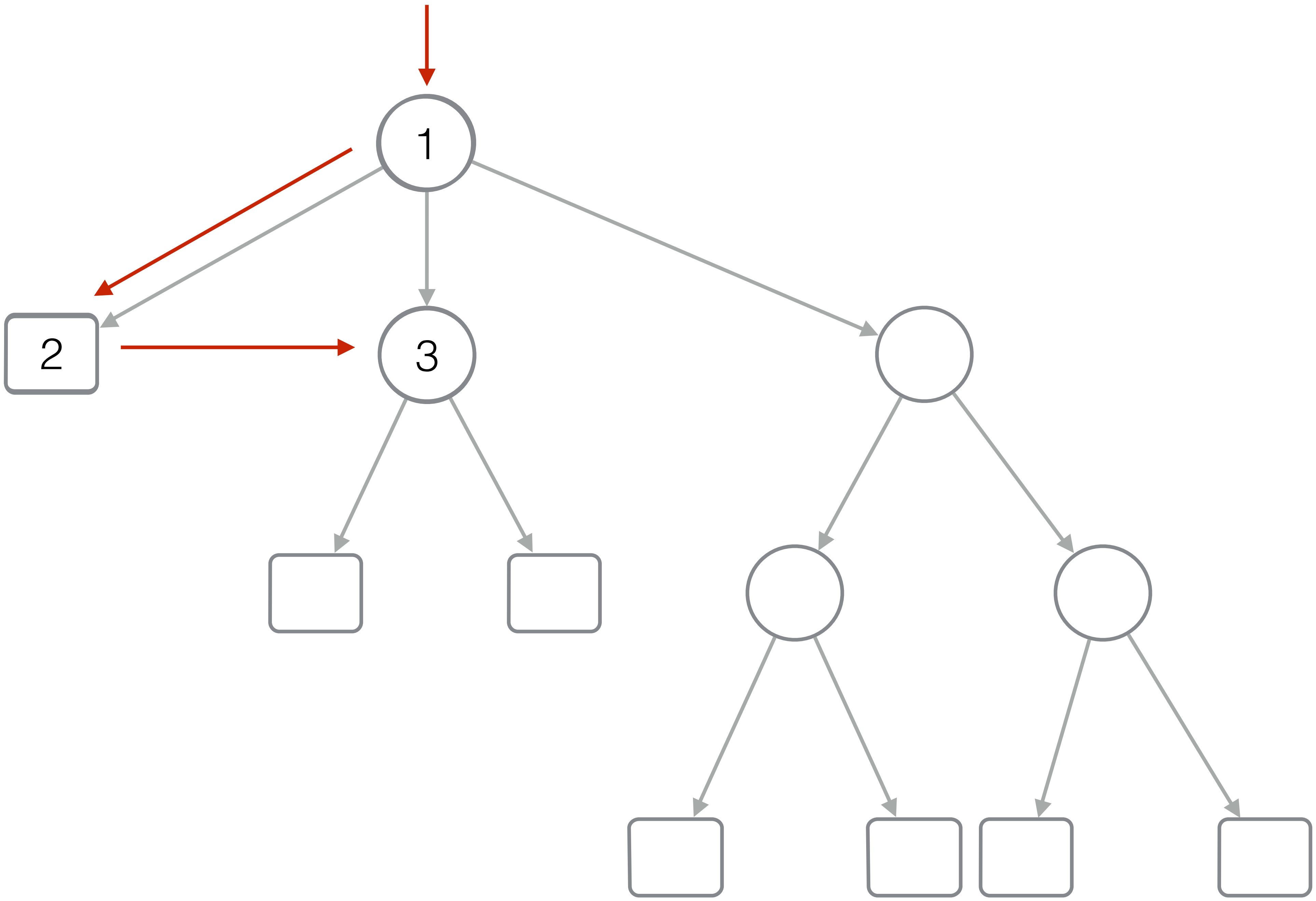
Succinct representation of trees (2)

[BP - Balanced parenthesis]



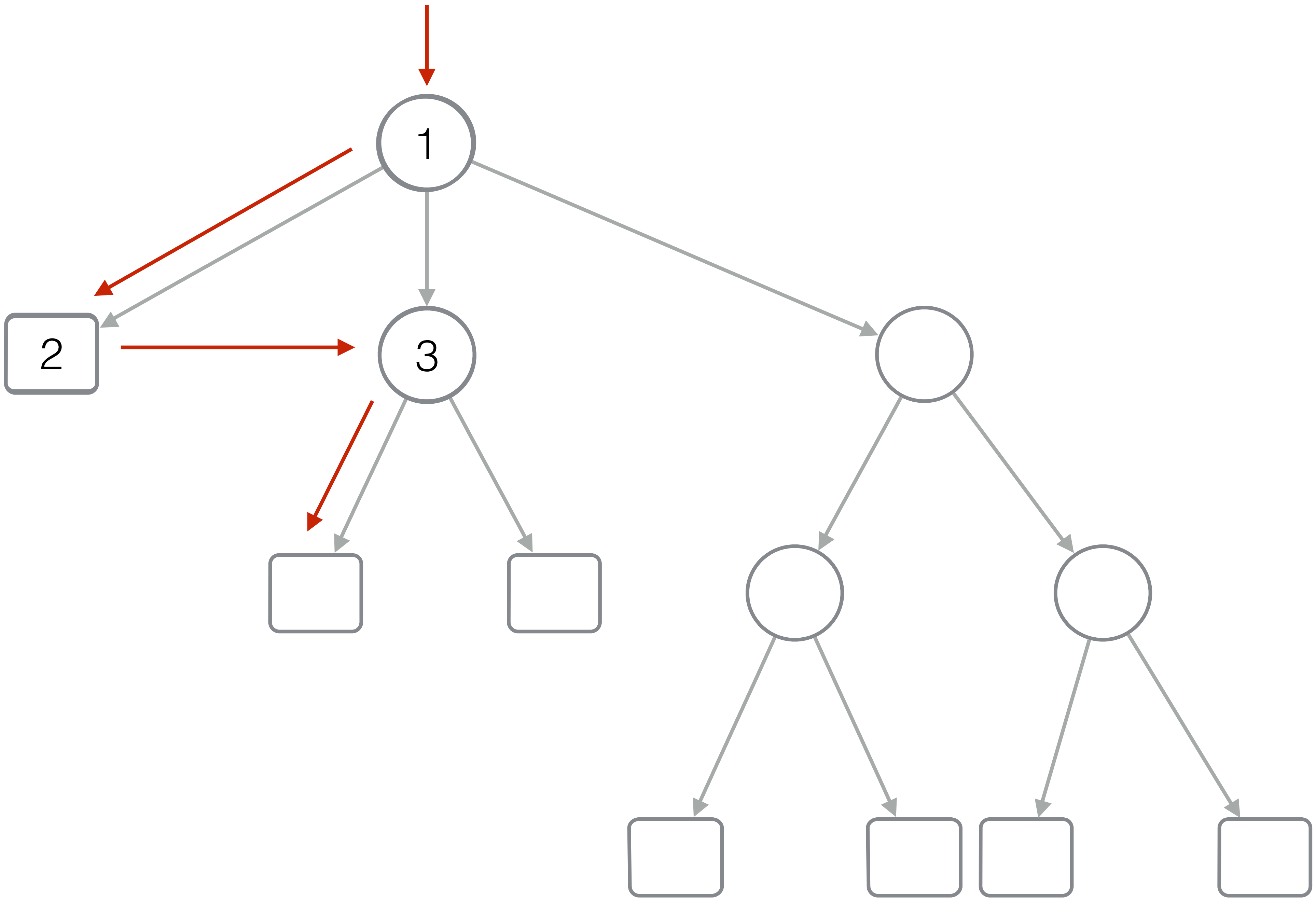
Succinct representation of trees (2)

[BP - Balanced parenthesis]



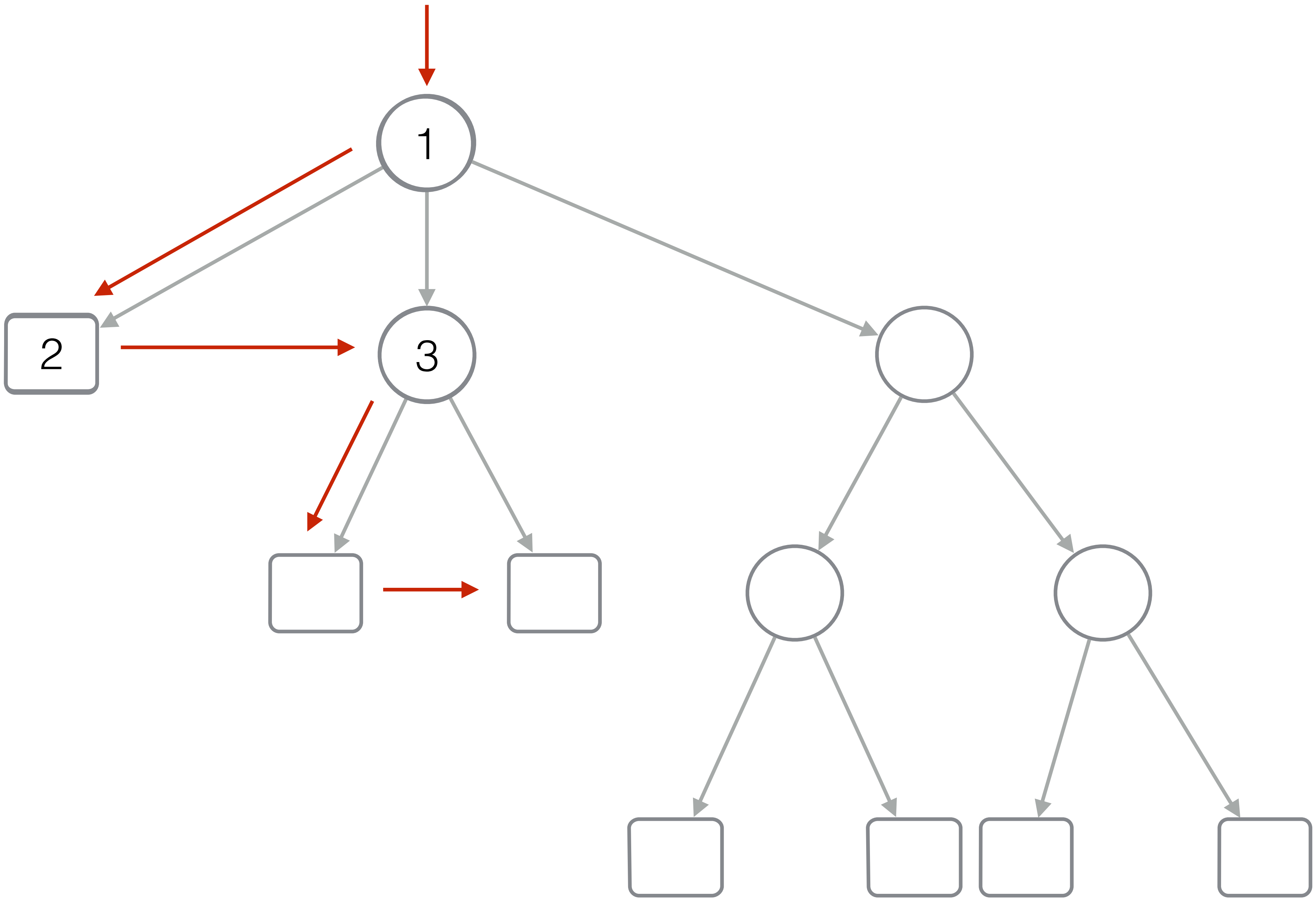
Succinct representation of trees (2)

[BP - Balanced parenthesis]



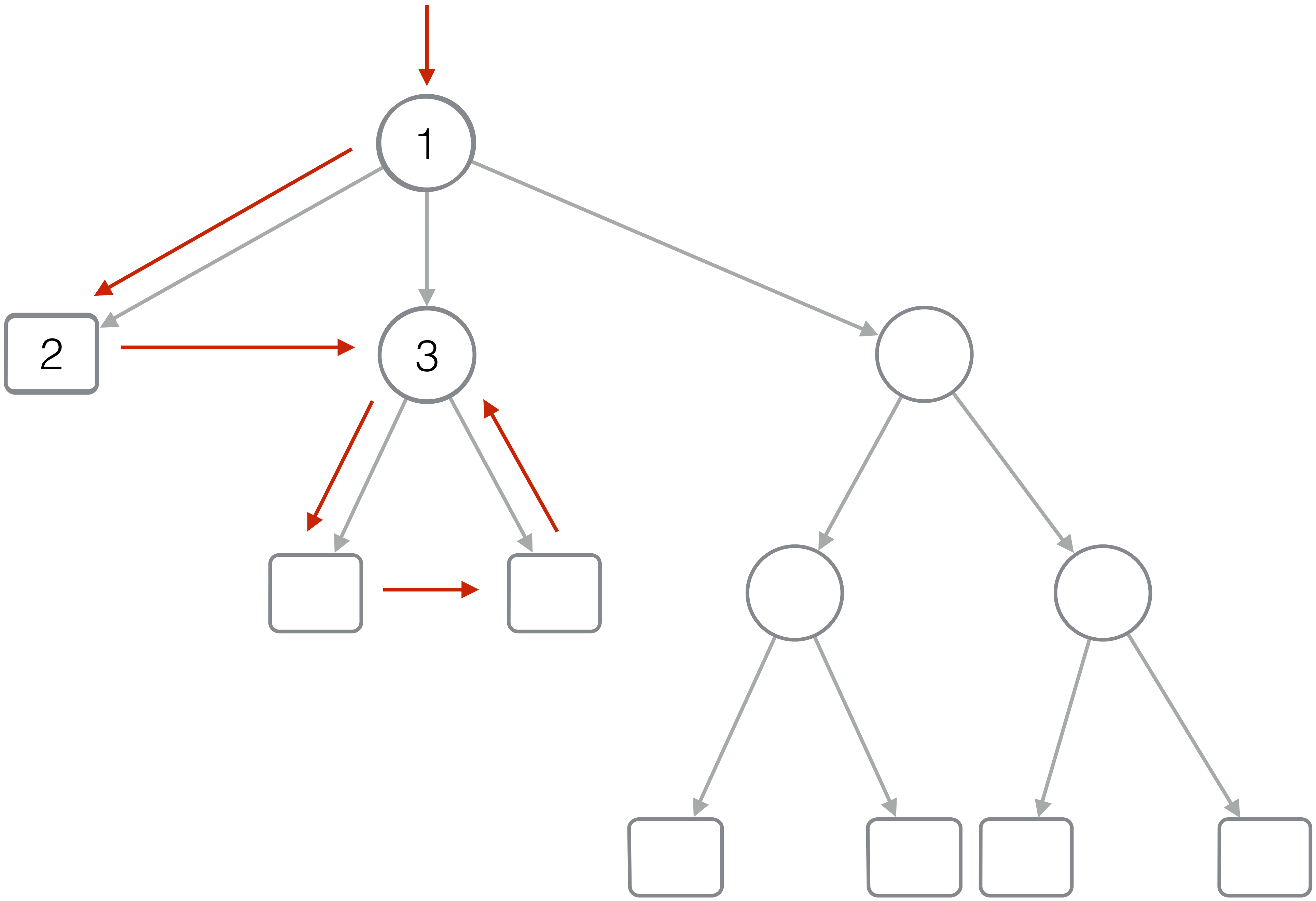
Succinct representation of trees (2)

[BP - Balanced parenthesis]



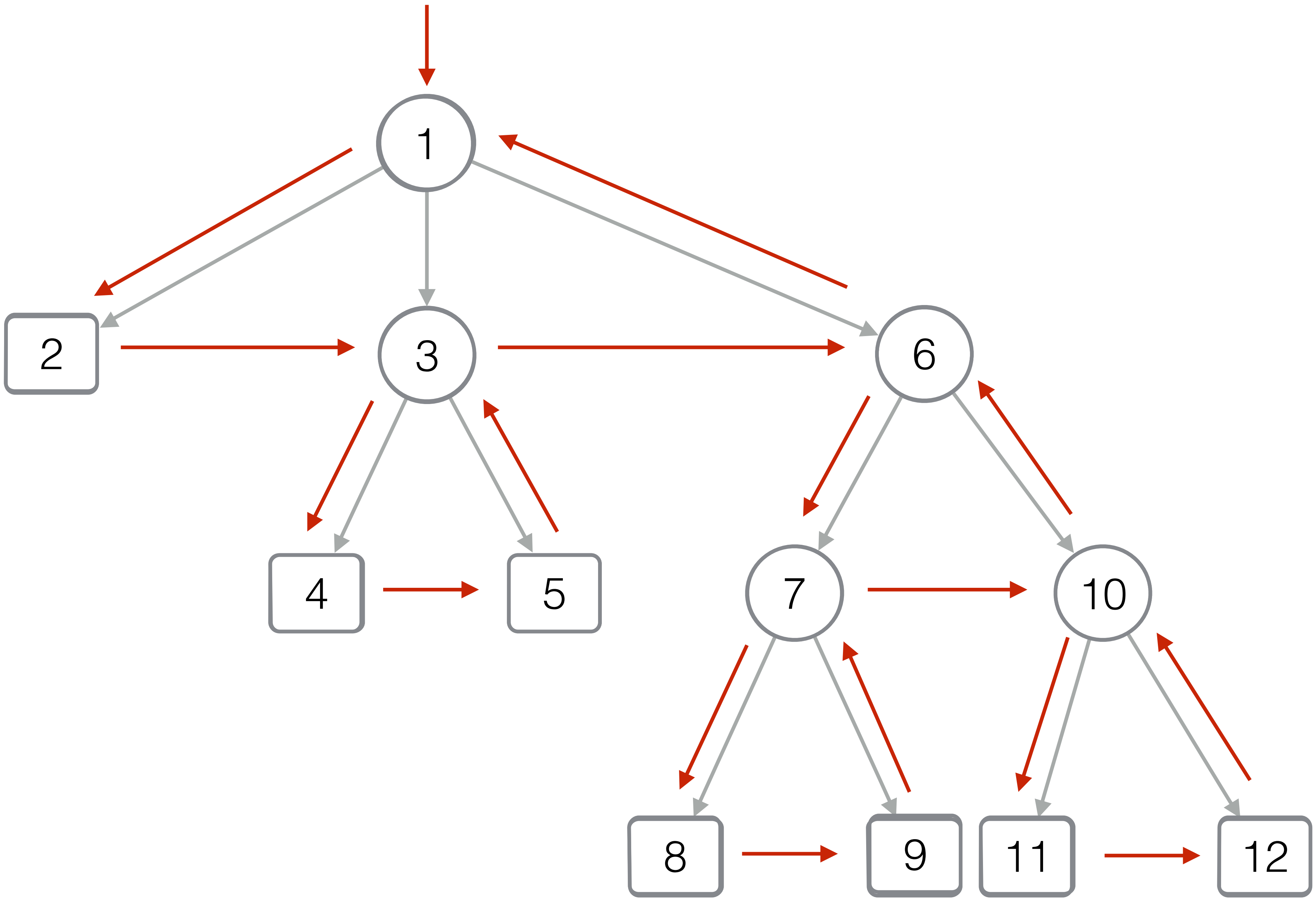
Succinct representation of trees (2)

[BP - Balanced parenthesis]



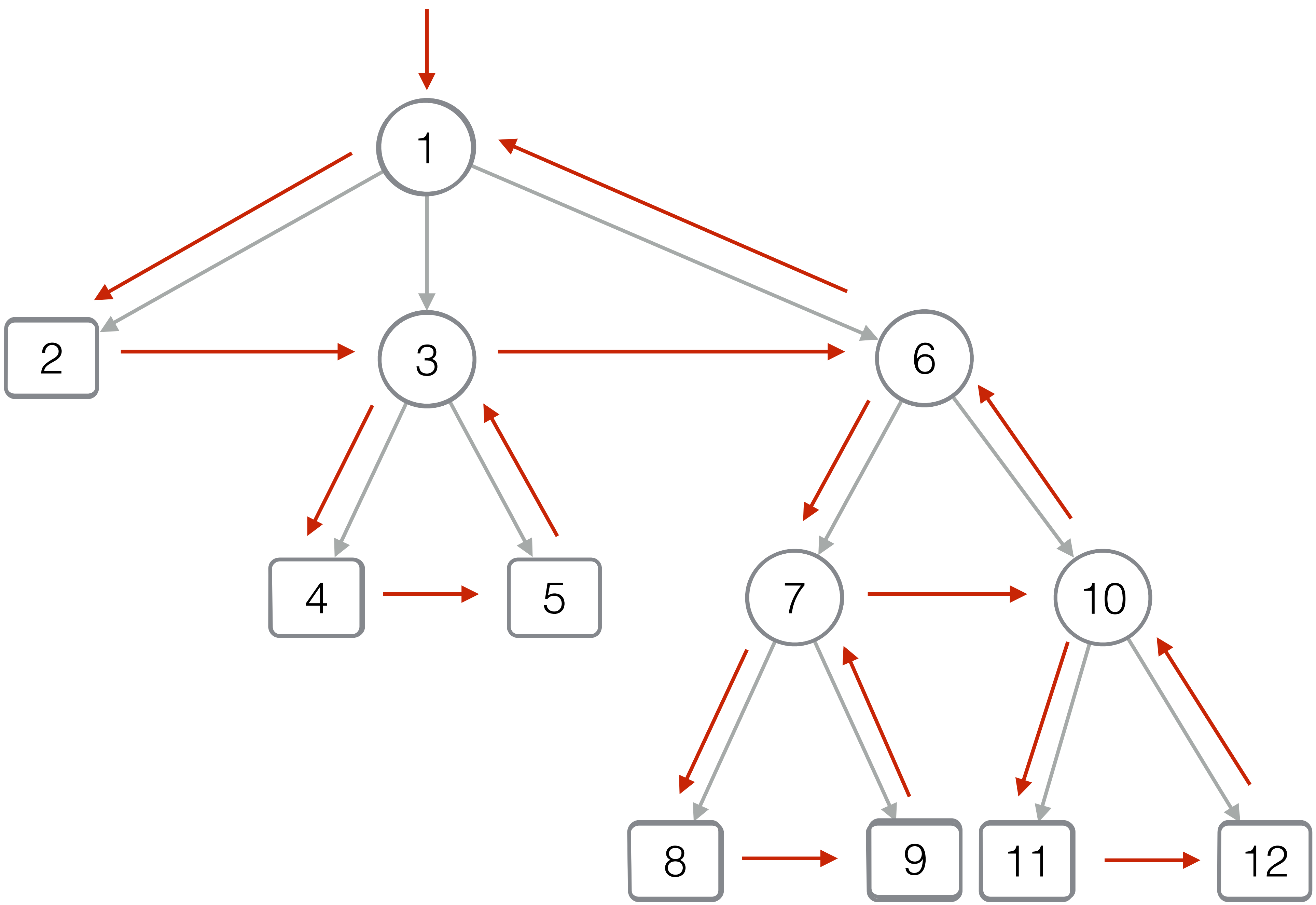
Succinct representation of trees (2)

[BP - Balanced parenthesis]



Succinct representation of trees (2)

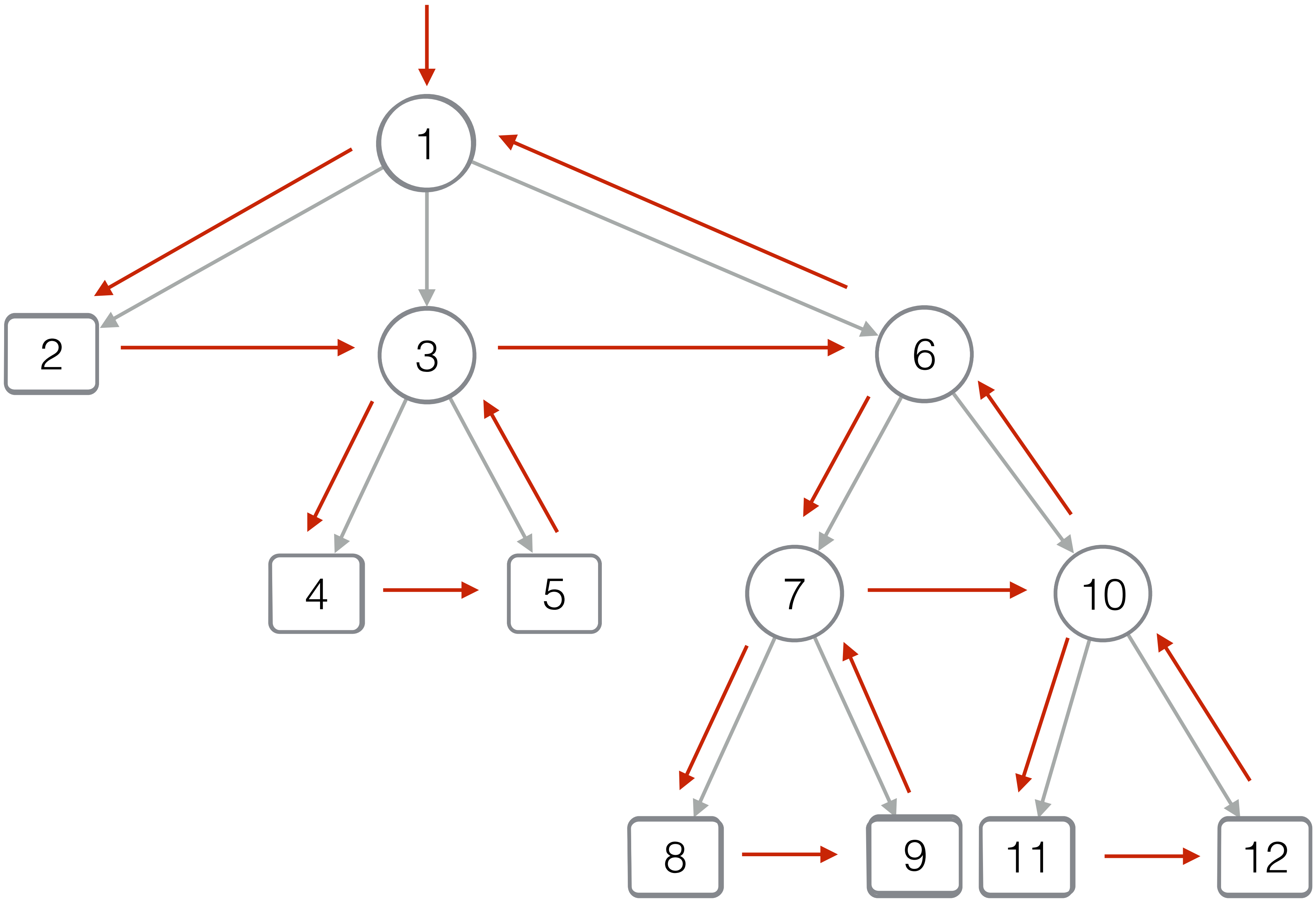
[BP - Balanced parenthesis]



B (( ) (( ) ( )) ((( ) ( )) (( ) ( ))))

Succinct representation of trees (2)

[BP - Balanced parenthesis]



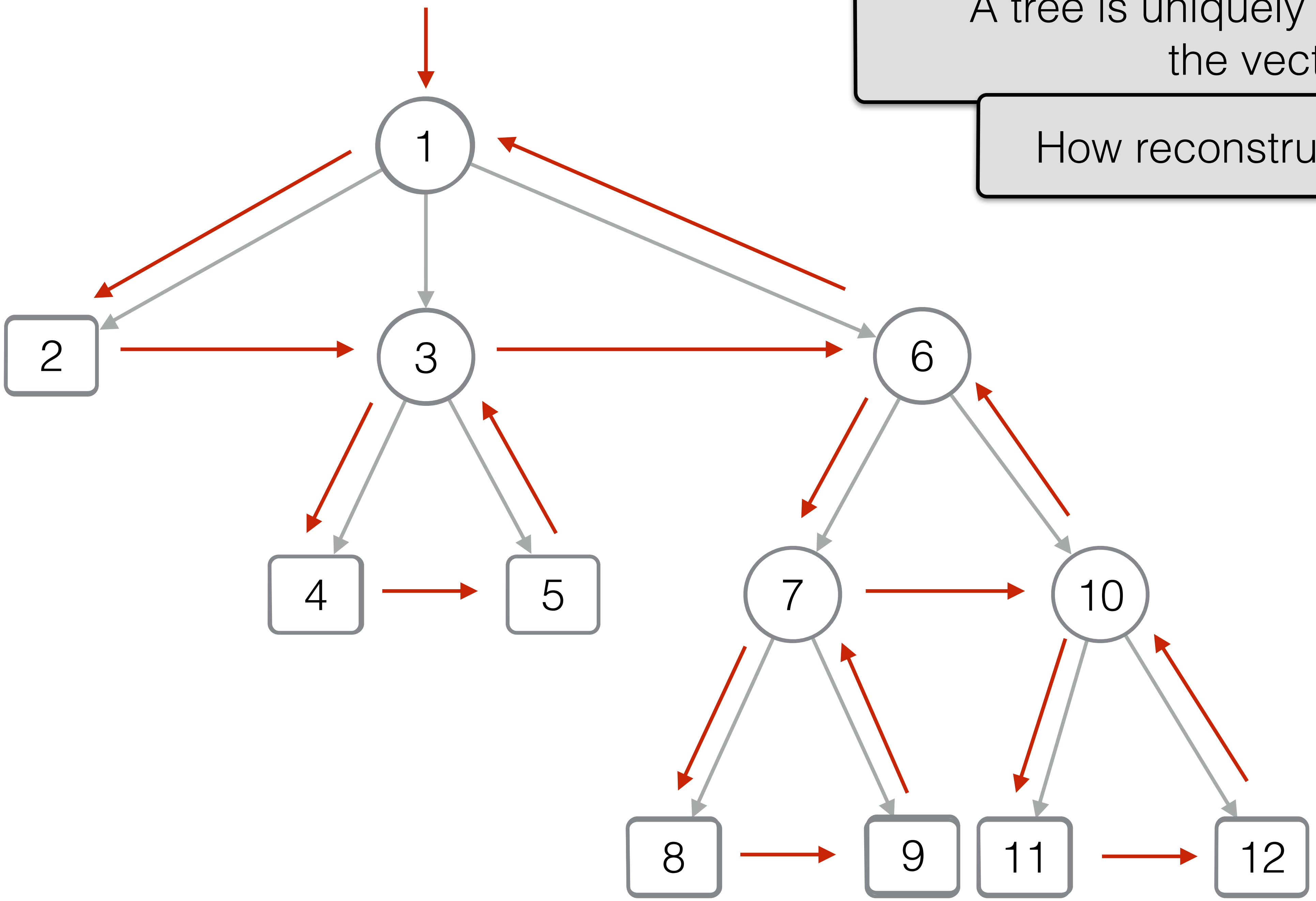
B	(	(	)	(	(	)	(	)	)	(	(	(	)	(	)	)	(	(	)	(	)	)	)	)
	1	2	2	3	4	4	5	5	3	6	7	8	8	9	9	7	10	11	11	12	12	10	6	1

Succinct representation of trees (2)

[BP - Balanced parenthesis]

A tree is uniquely determined by the vector B

How reconstruct the tree?

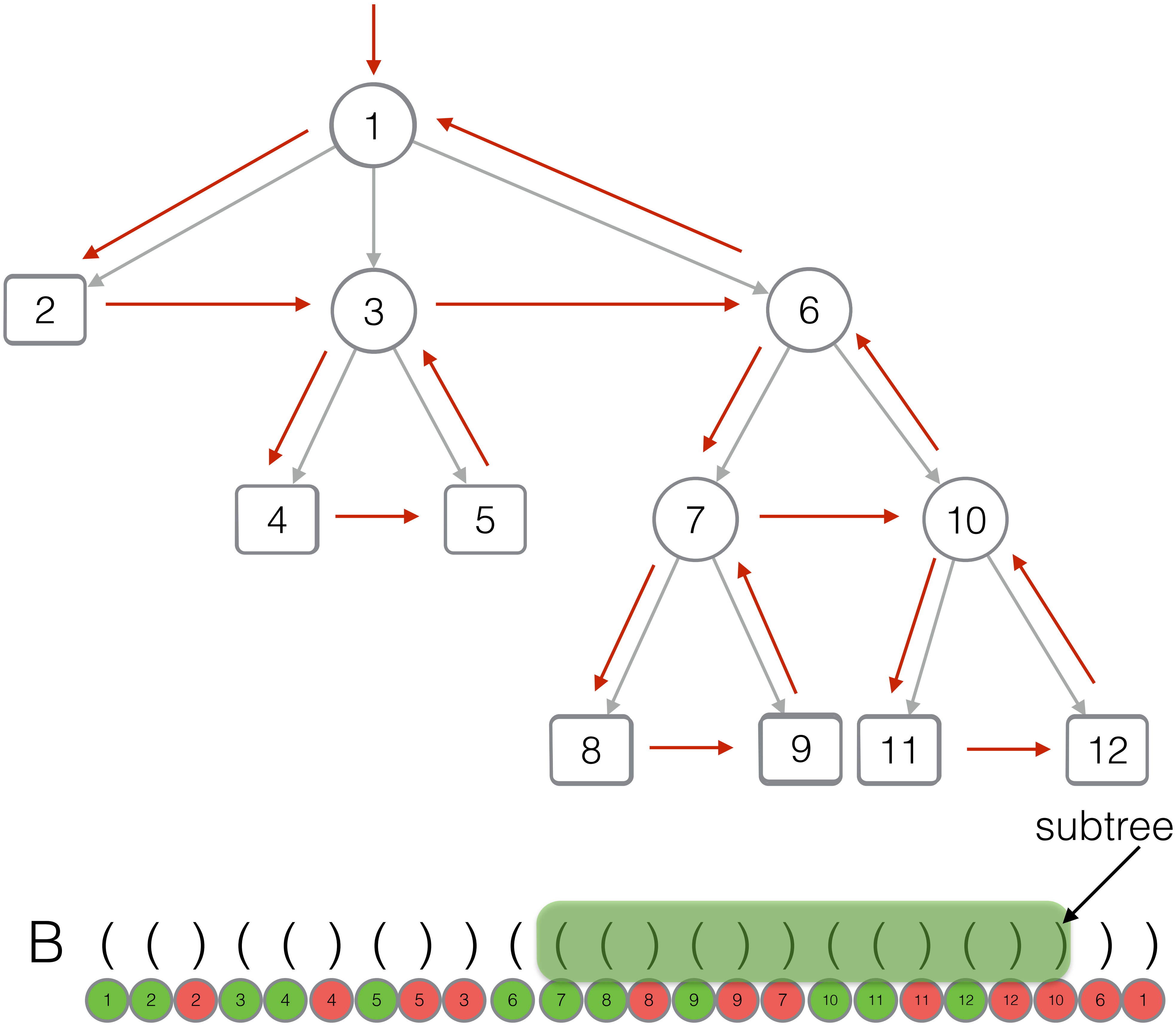


B (( ) (( ) ( )) ((( ) ( )) (( ) ( ))))

1 2 2 3 4 4 5 5 3 6 7 8 8 9 9 7 10 11 11 12 12 10 6 1

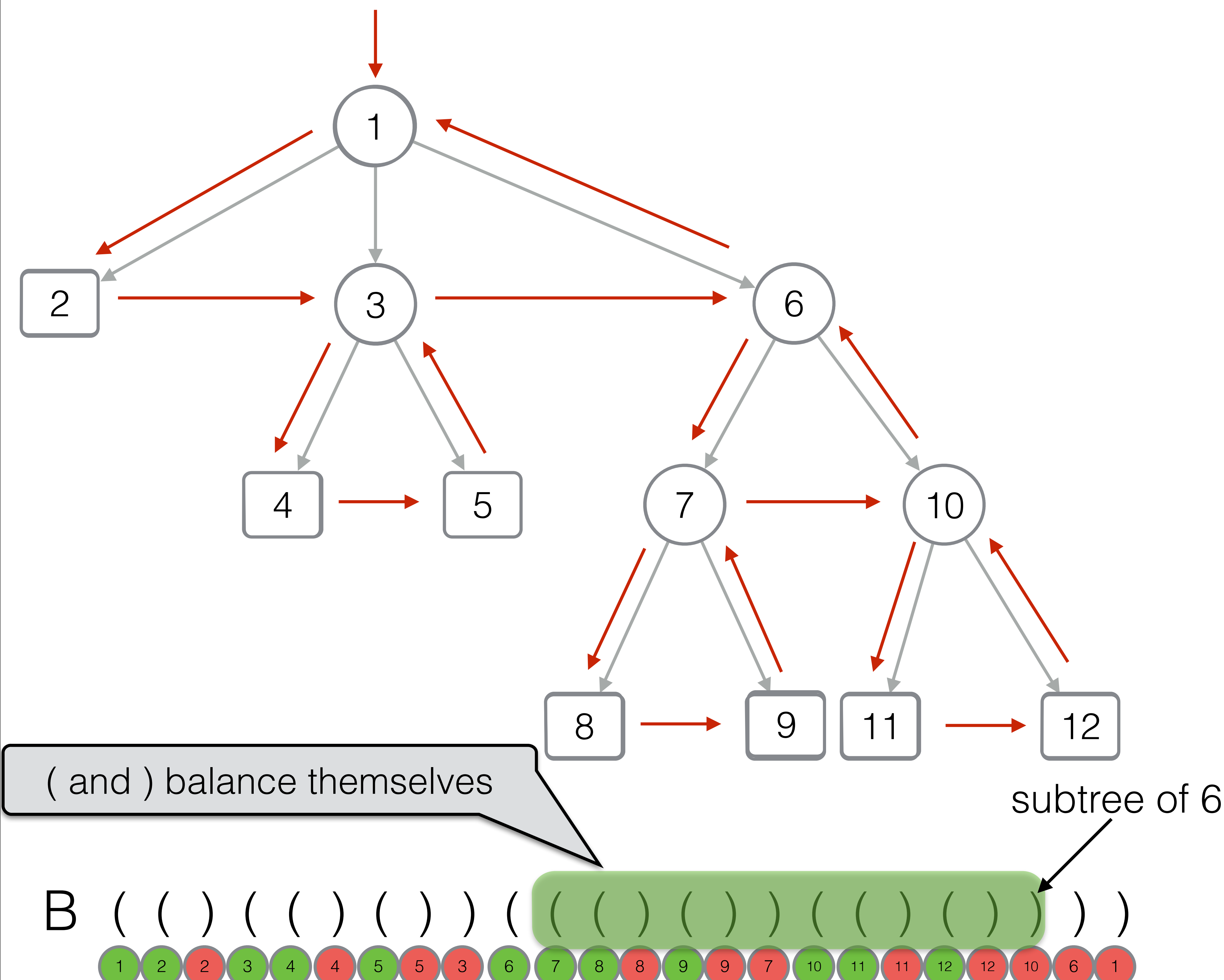
Succinct representation of trees (2)

[BP - Balanced parenthesis]



Succinct representation of trees (2)

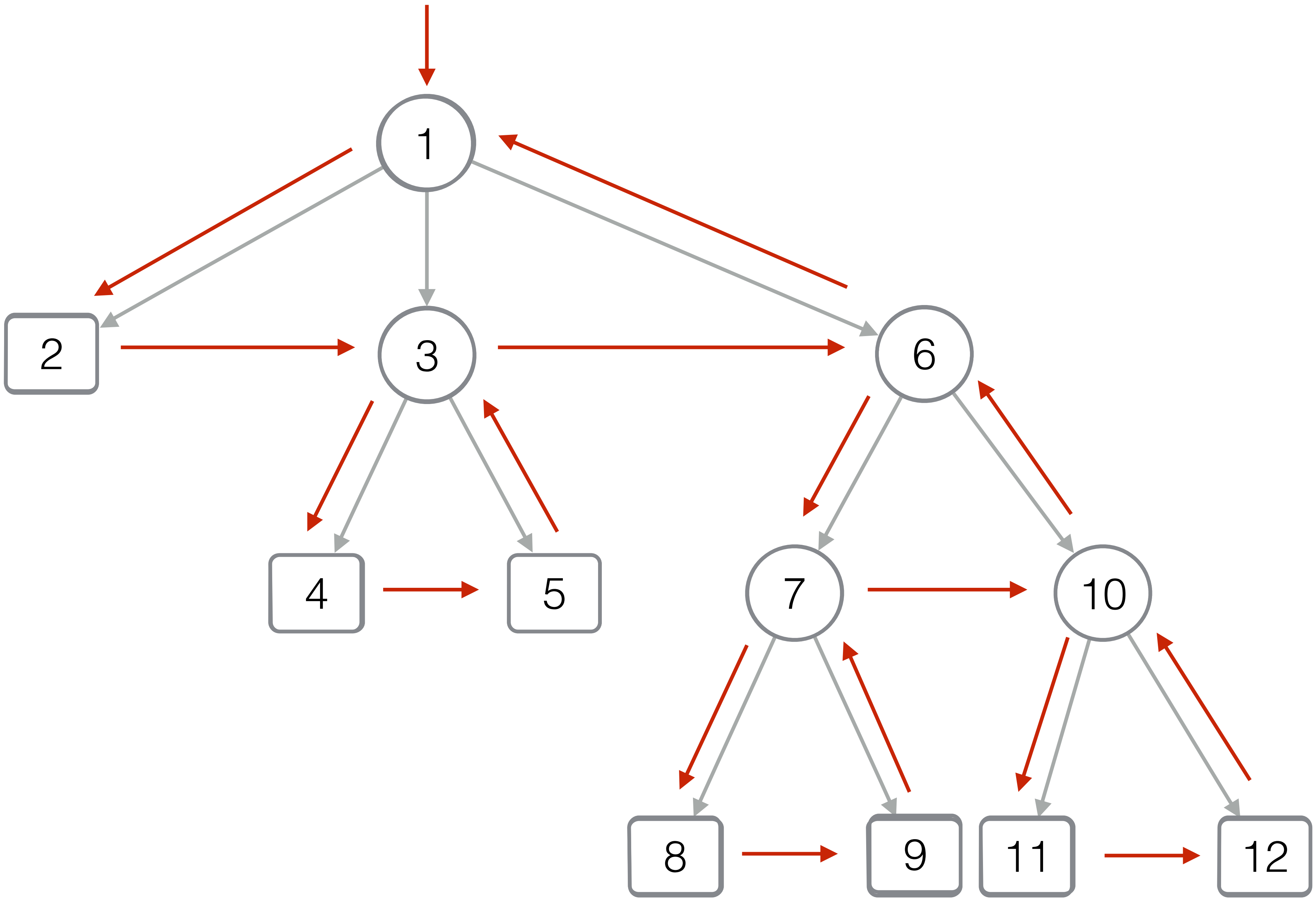
[BP - Balanced parenthesis]



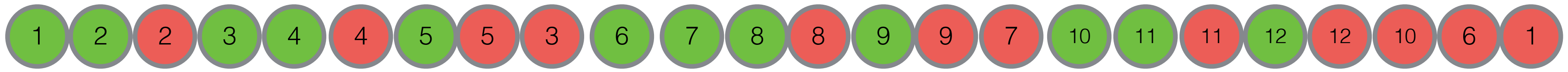
Succinct representation of trees (2)

[BP - Balanced parenthesis]

pos(x) =



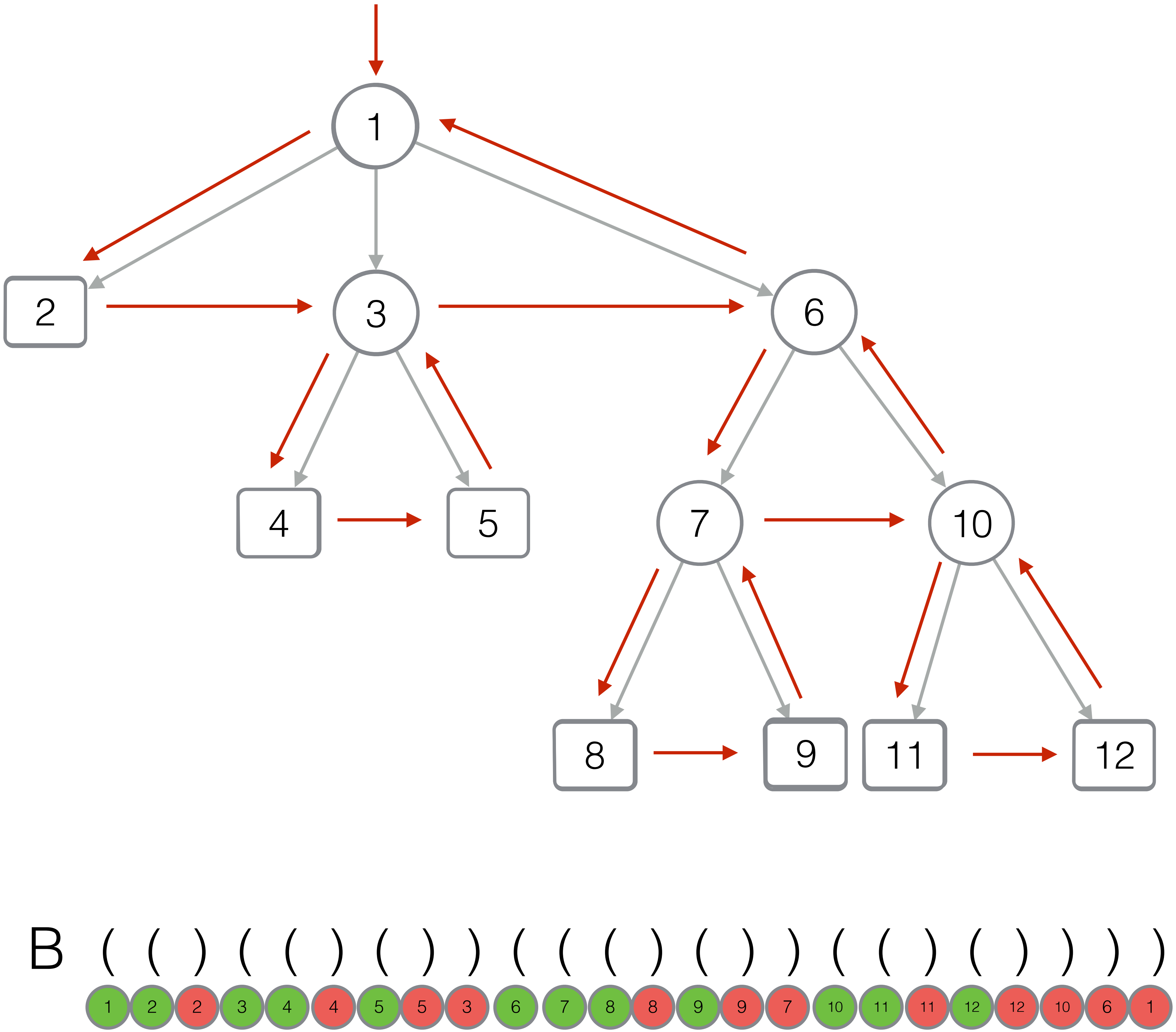
B (( ) (( ) ( )) ((( ) ( )) (( ) ( ))))



Succinct representation of trees (2)

[BP - Balanced parenthesis]

pos(x) = Select ((x)

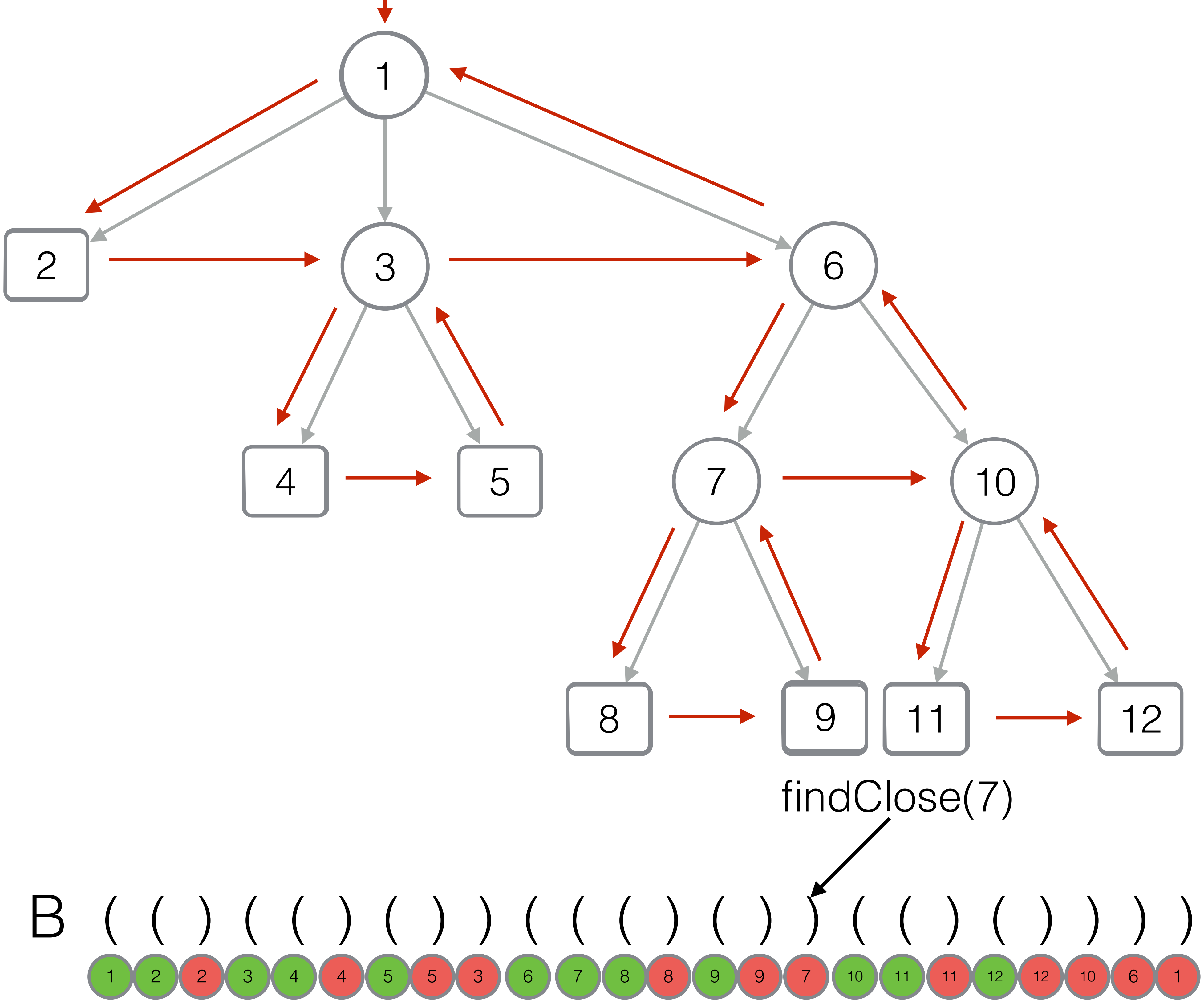


Succinct representation of trees (2)

[BP - Balanced parenthesis]

pos(x) = Select $_($ (x)

findClose(x) = returns the position of) matching x-th (



Succinct representation of trees (2)

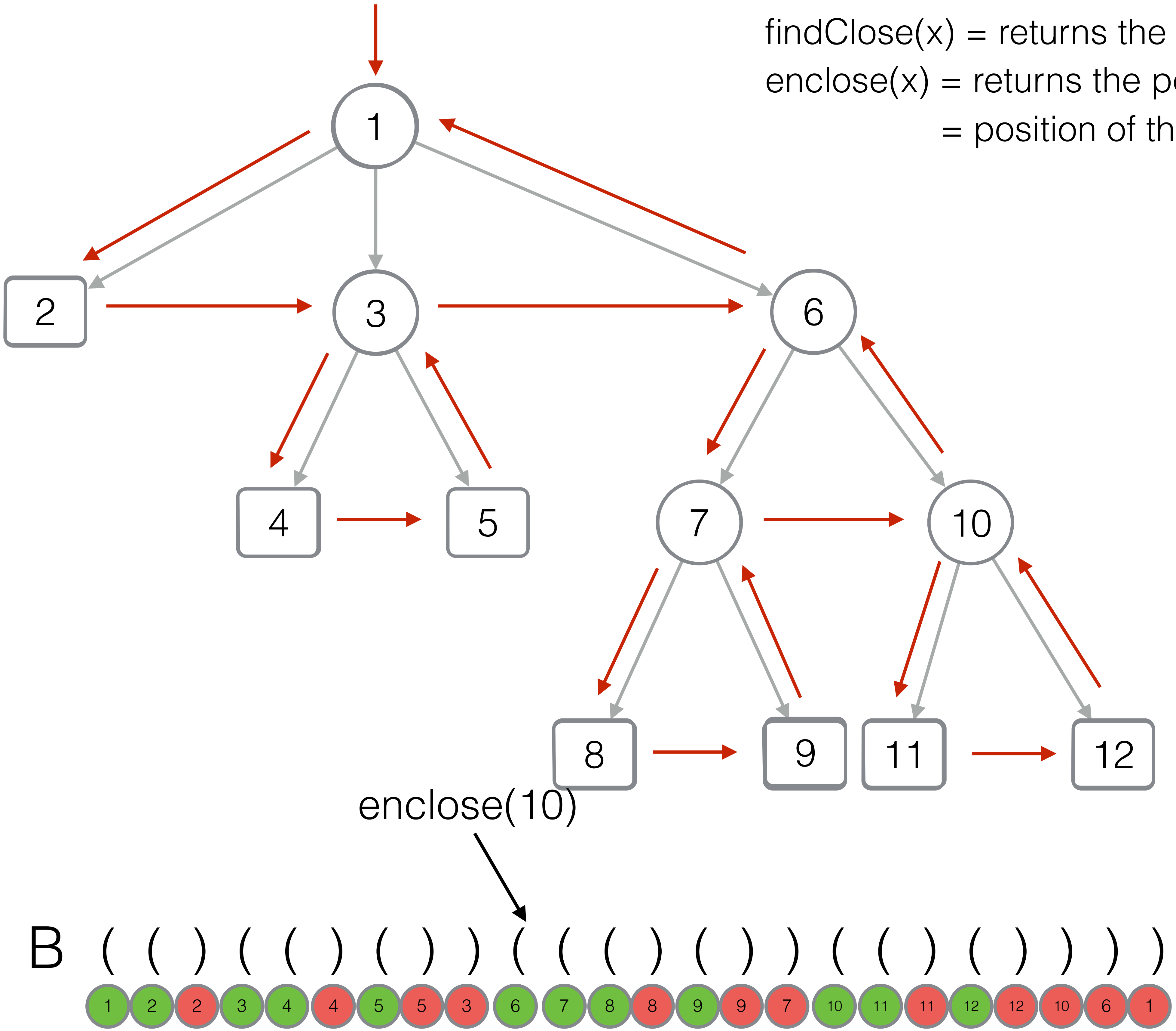
[BP - Balanced parenthesis]

pos(x) = Select $_($ (x)

findClose(x) = returns the position of) matching x-th (

enclose(x) = returns the position of (enclosing x-th (

= position of the parent of x in B



Succinct representation of trees (2)

[BP - Balanced parenthesis]

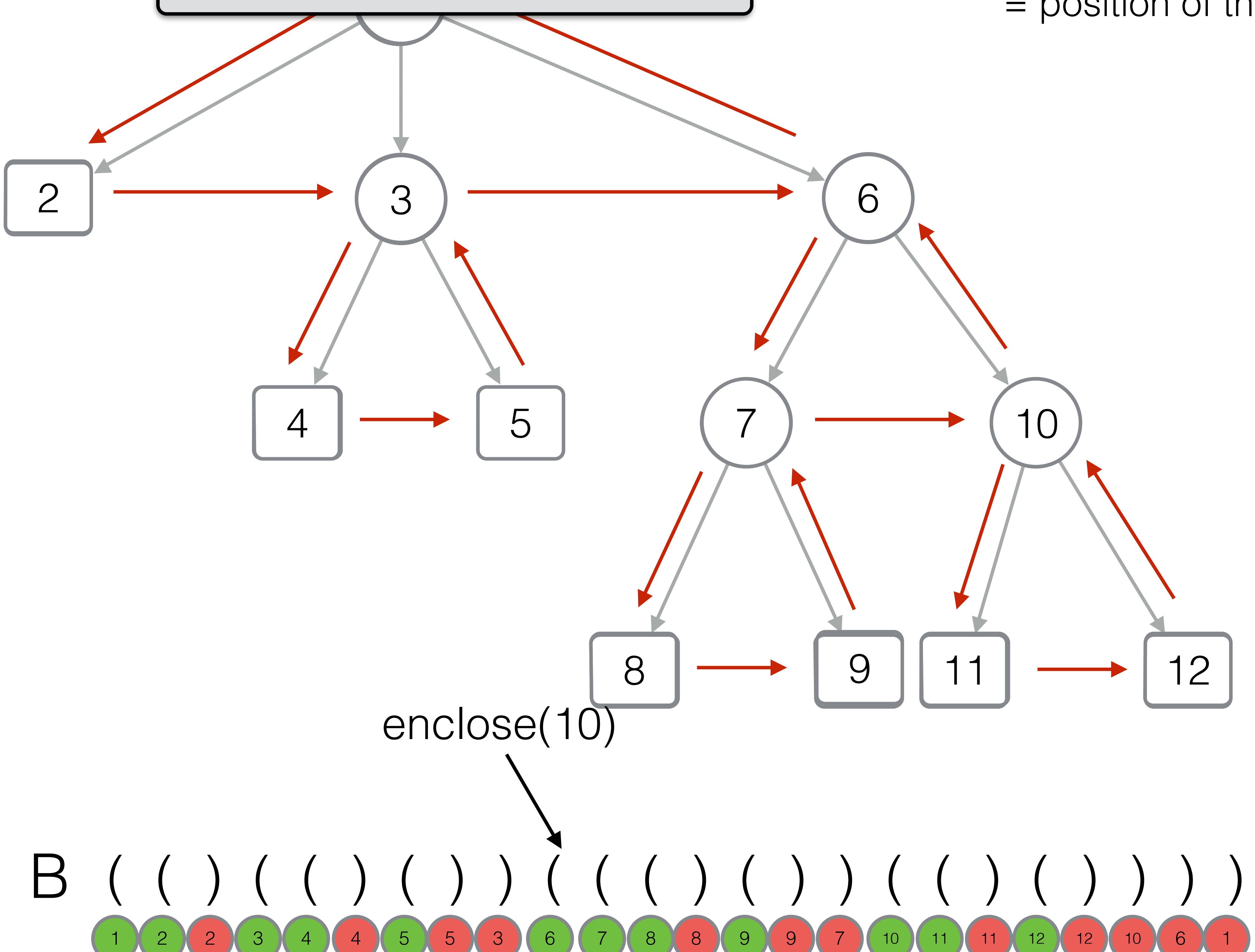
They can be implemented in O(1) time.

pos(x) = $\text{Select}_{(}(x)$

findClose(x) = returns the position of) matching x-th (

enclose(x) = returns the position of (enclosing x-th (

= position of the parent of x in B



Succinct representation of trees (2)

[BP - Balanced parenthesis]

They can be implemented in O(1) time.

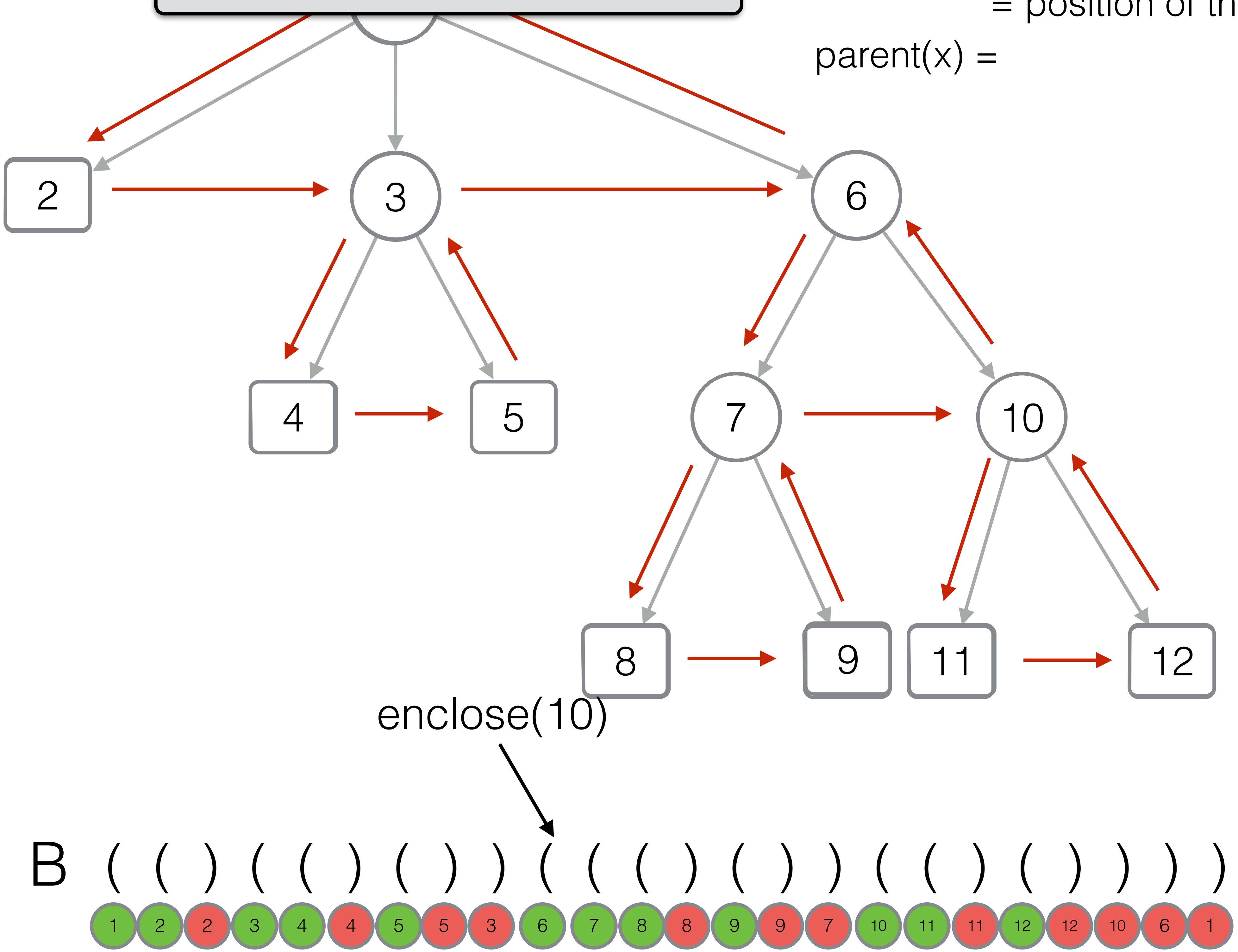
pos(x) = Select $_($ (x)

findClose(x) = returns the position of) matching x-th (

enclose(x) = returns the position of (enclosing x-th (

= position of the parent of x in B

parent(x) =



Succinct representation of trees (2)

[BP - Balanced parenthesis]

They can be implemented in O(1) time.

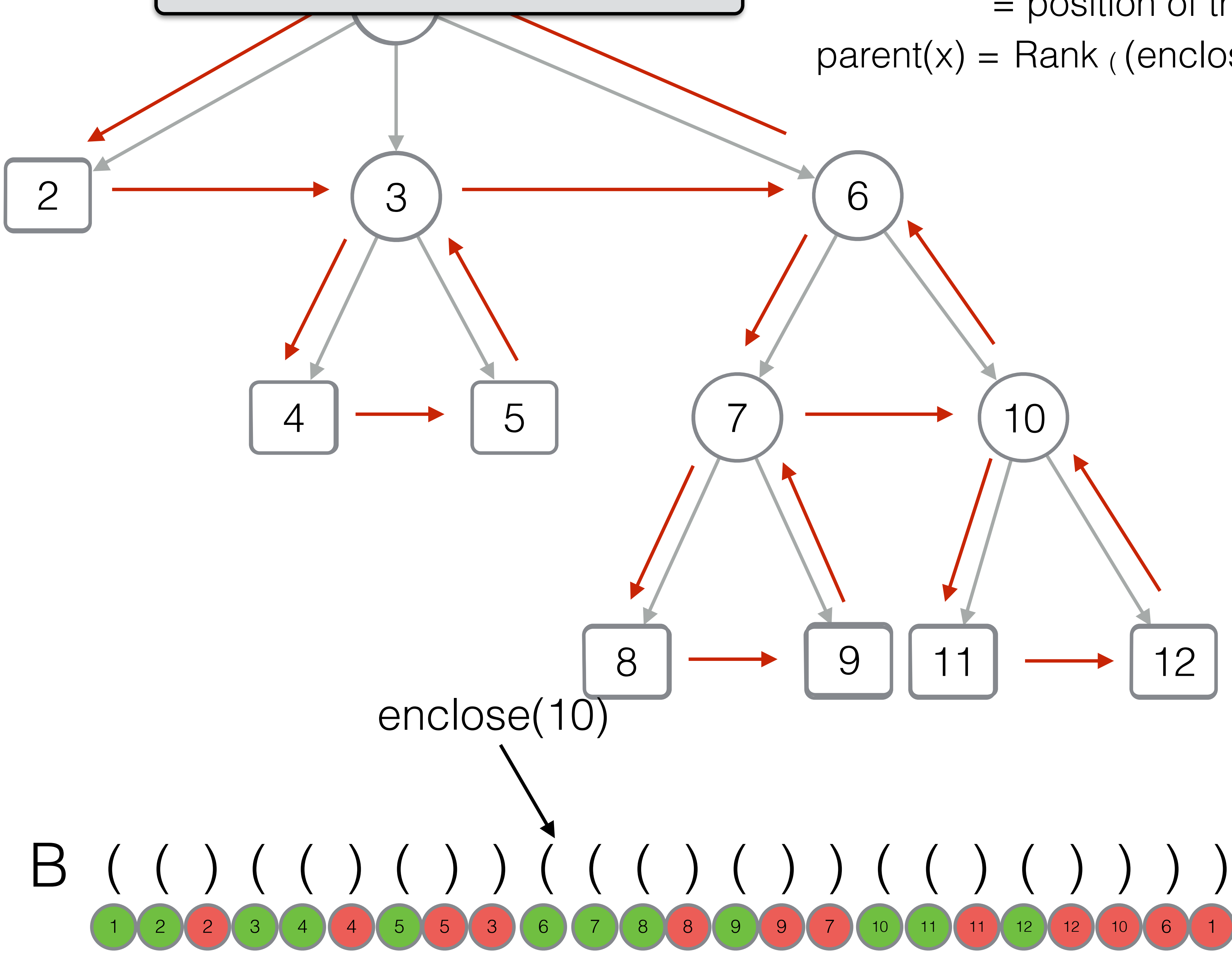
pos(x) = Select $_($ (x)

findClose(x) = returns the position of) matching x-th (

enclose(x) = returns the position of (enclosing x-th (

= position of the parent of x in B

parent(x) = Rank $_($ (enclose(x))



Succinct representation of trees (2)

[BP - Balanced parenthesis]

They can be implemented in O(1) time.

pos(x) = Select ((x)

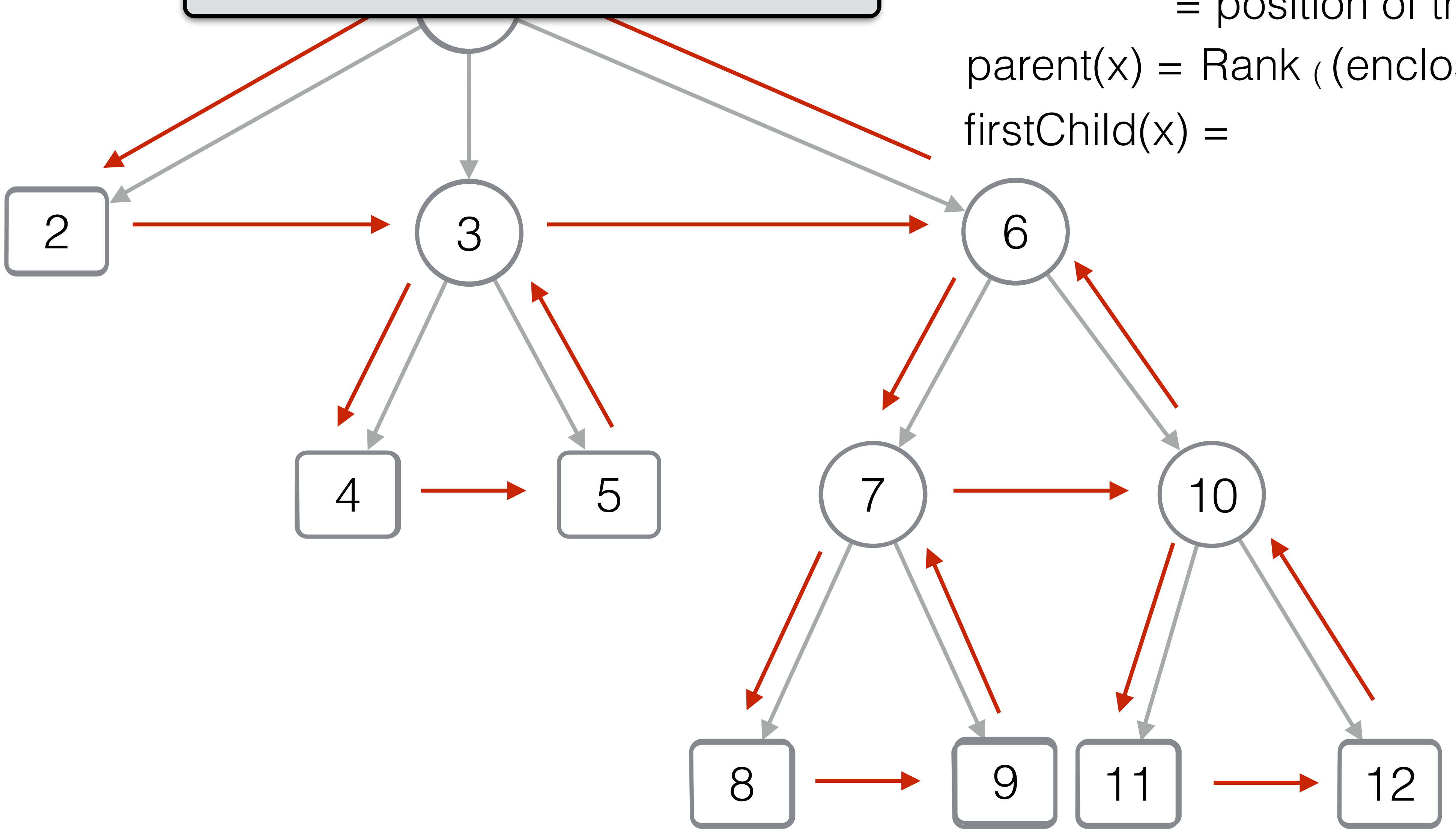
findClose(x) = returns the position of) matching x-th (

enclose(x) = returns the position of (enclosing x-th (

= position of the parent of x in B

parent(x) = Rank ((enclose(x))

firstChild(x) =



B (( ) (( ) ( )) ((( ) ( )) (( ) ( ))))

1 2 2 3 4 4 5 5 3 6 7 8 8 9 9 7 10 11 11 12 12 10 6 1

Succinct representation of trees (2)

[BP - Balanced parenthesis]

They can be implemented in O(1) time.

pos(x) = Select ((x)

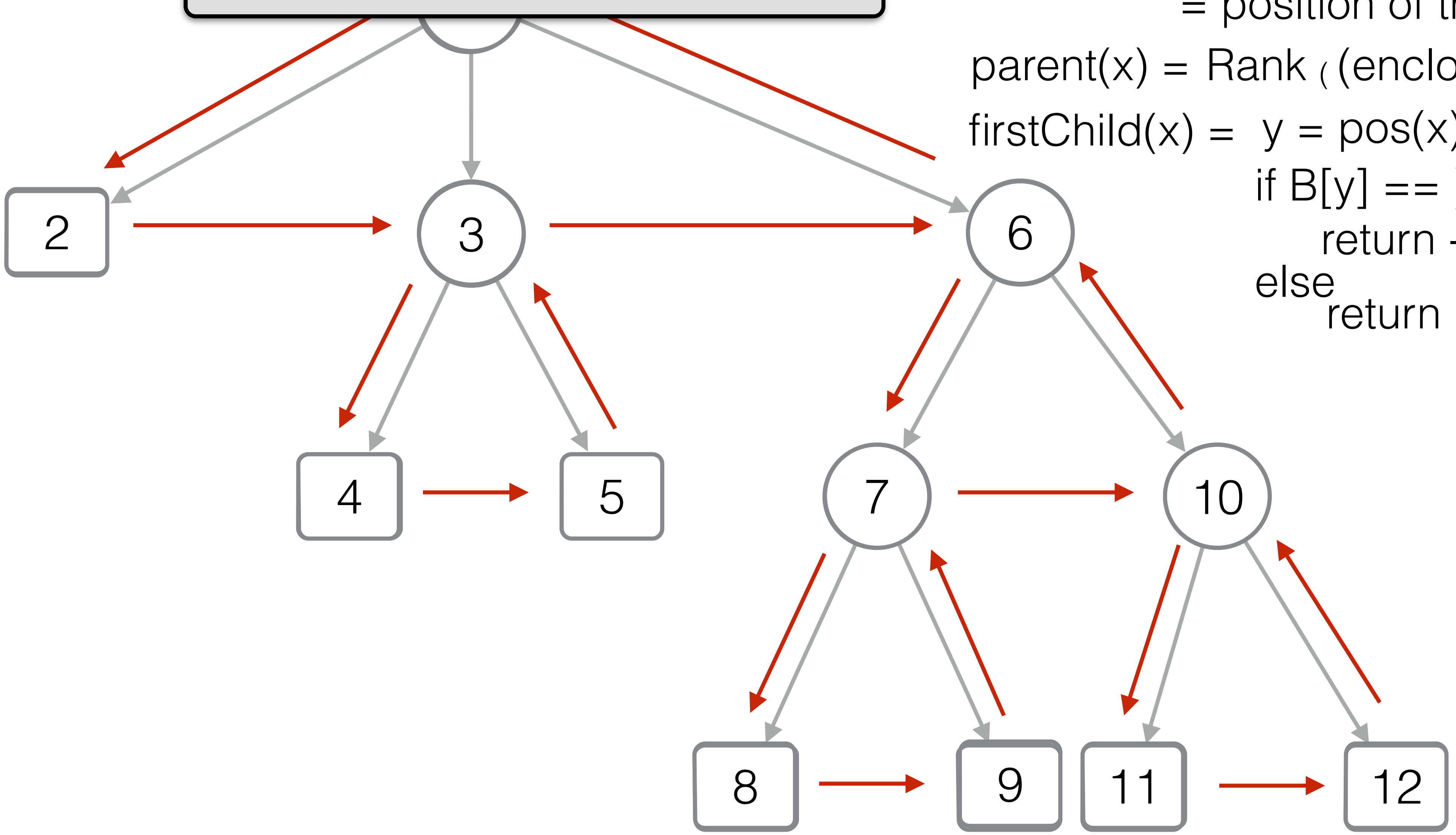
findClose(x) = returns the position of) matching x-th (

enclose(x) = returns the position of (enclosing x-th (

= position of the parent of x in B

parent(x) = Rank ((enclose(x))

```
firstChild(x) =  y = pos(x)+1
                 if B[y] == )
                      return -1      // is a leaf
                 else
                      return Rank ( (y)
```



B (( ) (( ) ( )) ((( ) ( )) (( ) ( ))))

1 2 2 3 4 4 5 5 3 6 7 8 8 9 9 7 10 11 11 12 12 10 6 1

Succinct representation of trees (2)

[BP - Balanced parenthesis]

They can be implemented in O(1) time.

pos(x) = Select ((x)

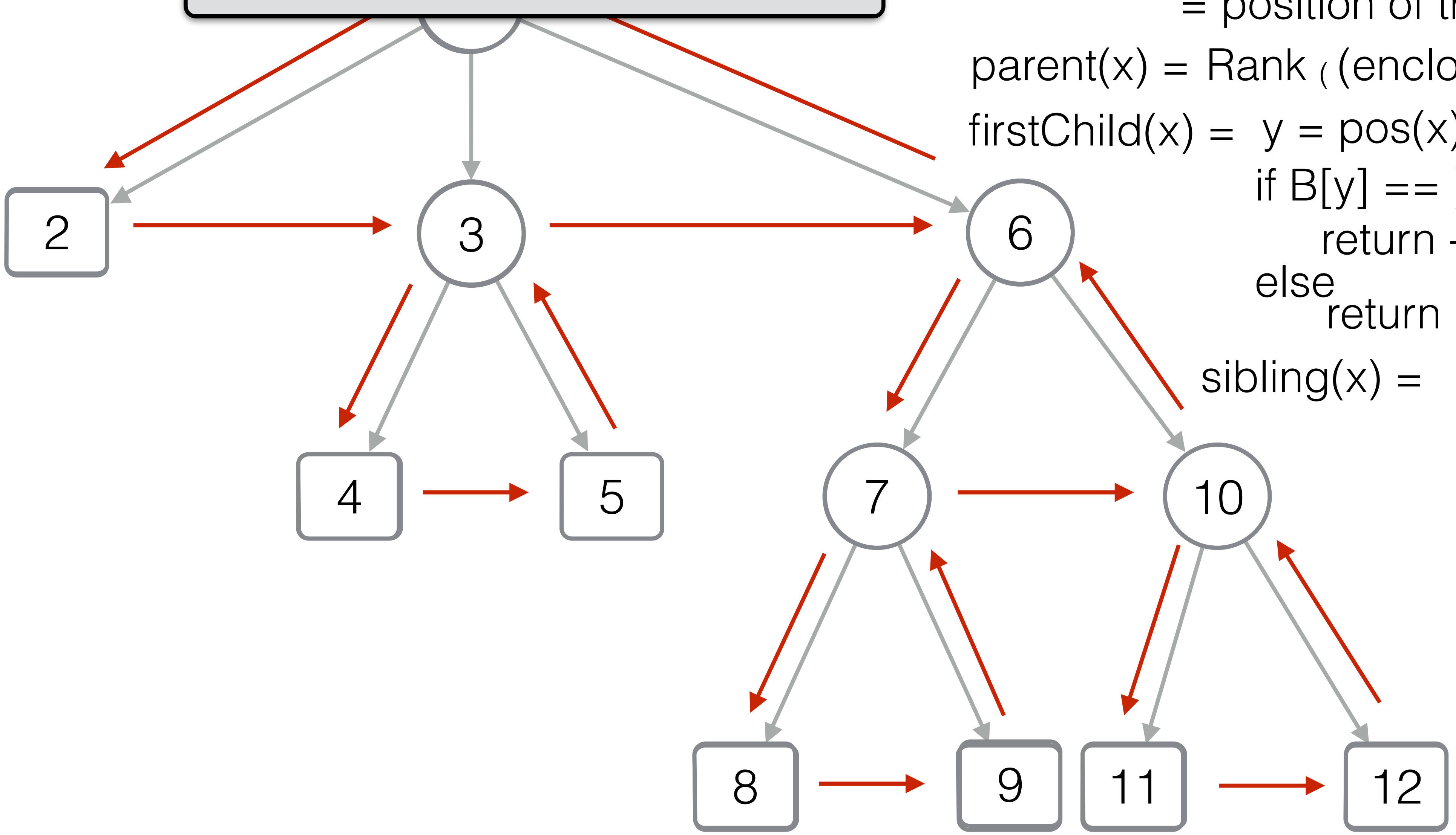
findClose(x) = returns the position of) matching x-th (

enclose(x) = returns the position of (enclosing x-th (
= position of the parent of x in B

parent(x) = Rank ((enclose(x))

```
firstChild(x) = y = pos(x)+1
                if B[y] == )
                    return -1     // is a leaf
                else
                    return Rank ( (y)
```

sibling(x) =



B	(	(	)	(	(	)	(	)	)	(	(	(	)	(	)	)	(	(	)	(	)	)	)	)
	1	2	2	3	4	4	5	5	3	6	7	8	8	9	9	7	10	11	11	12	12	10	6	1

Succinct representation of trees (2)

[BP - Balanced parenthesis]

They can be implemented in O(1) time.

pos(x) = Select$_($(x)

findClose(x) = returns the position of) matching x-th (

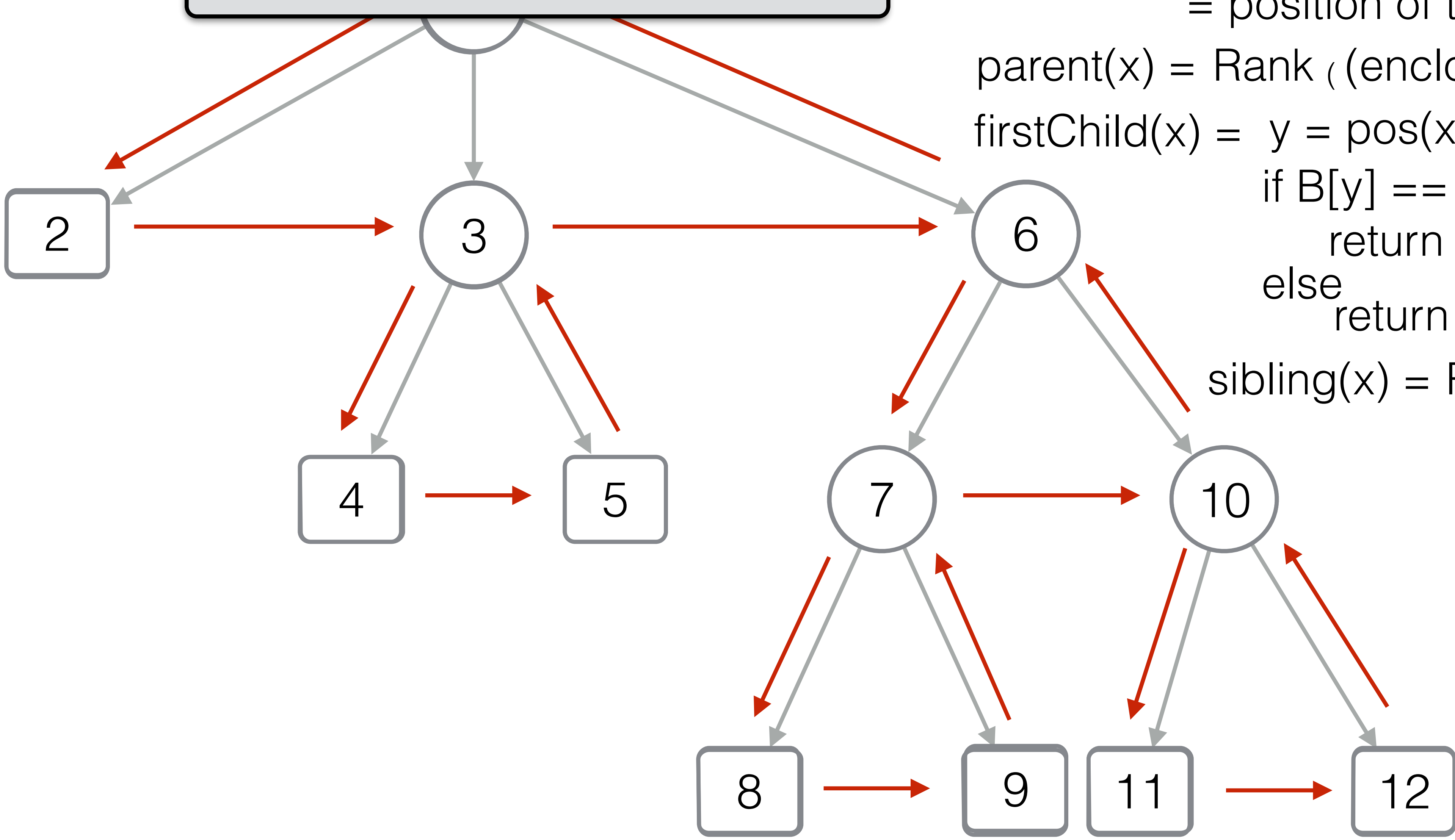
enclose(x) = returns the position of (enclosing x-th (

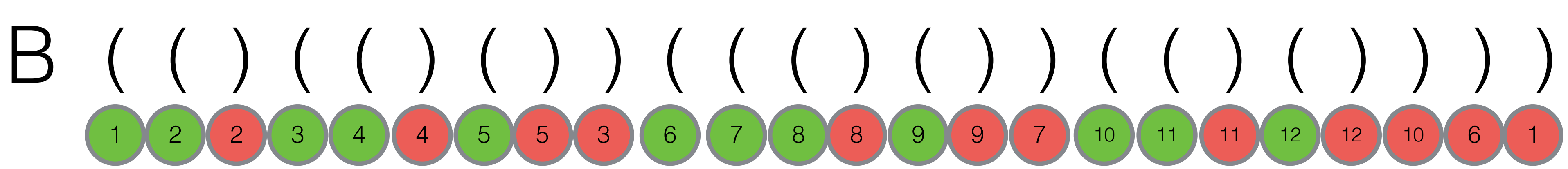
= position of the parent of x in B

parent(x) = Rank$_($(enclose(x))

firstChild(x) = y = pos(x)+1
if B[y] ==)
return -1 // is a leaf
else
return Rank$_($(y)

sibling(x) = Rank$_($(findClose(x)+1) (if any)





Succinct representation of trees (2)

[BP - Balanced parenthesis]

They can be implemented in O(1) time.

pos(x) = Select $_($ (x)

findClose(x) = returns the position of) matching x-th (

enclose(x) = returns the position of (enclosing x-th (

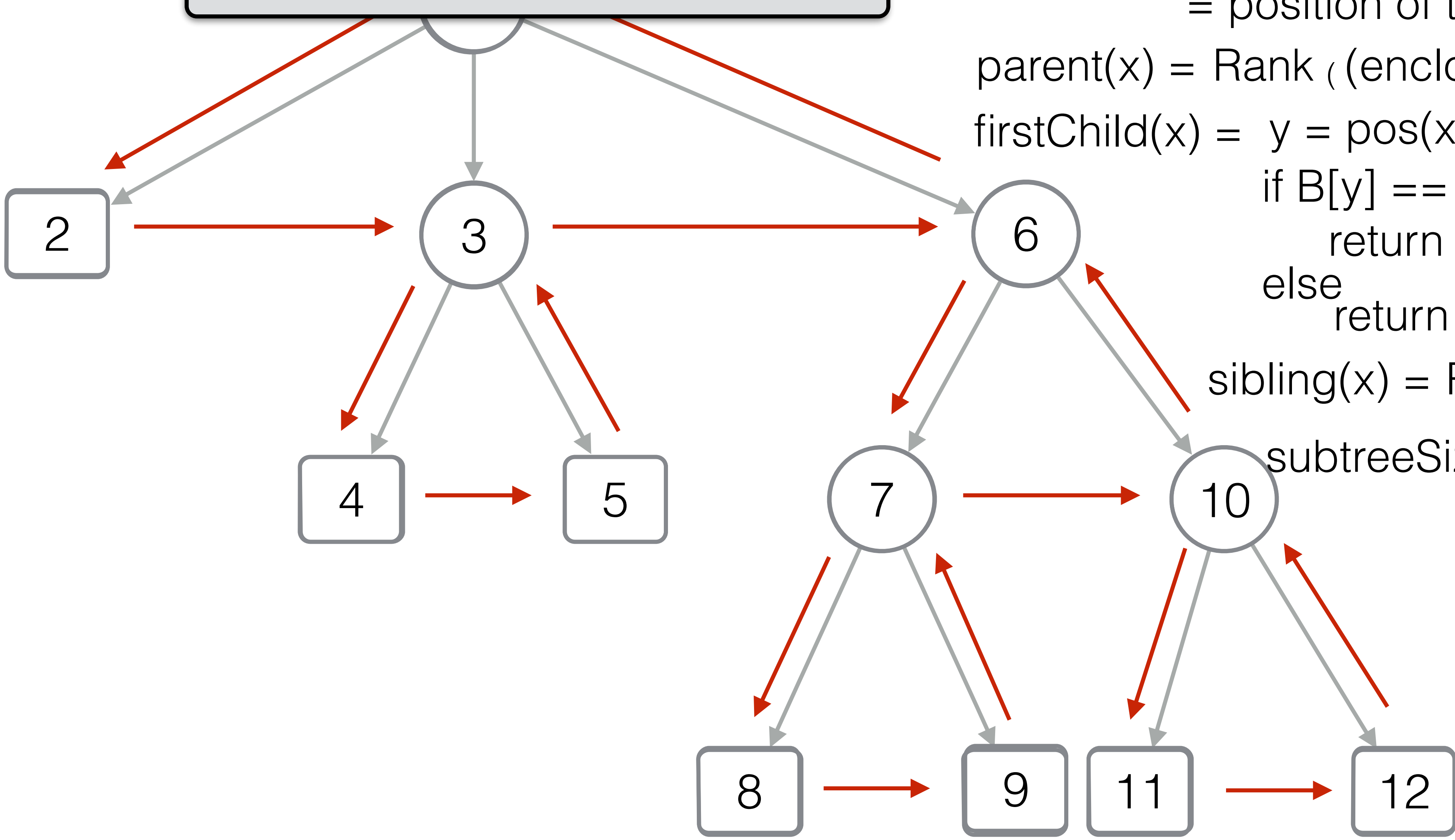
= position of the parent of x in B

parent(x) = Rank $_($ (enclose(x))

firstChild(x) = y = pos(x)+1
if B[y] ==)
return -1 // is a leaf
else
return Rank $_($ (y)

sibling(x) = Rank $_($ (findClose(x)+1) (if any)

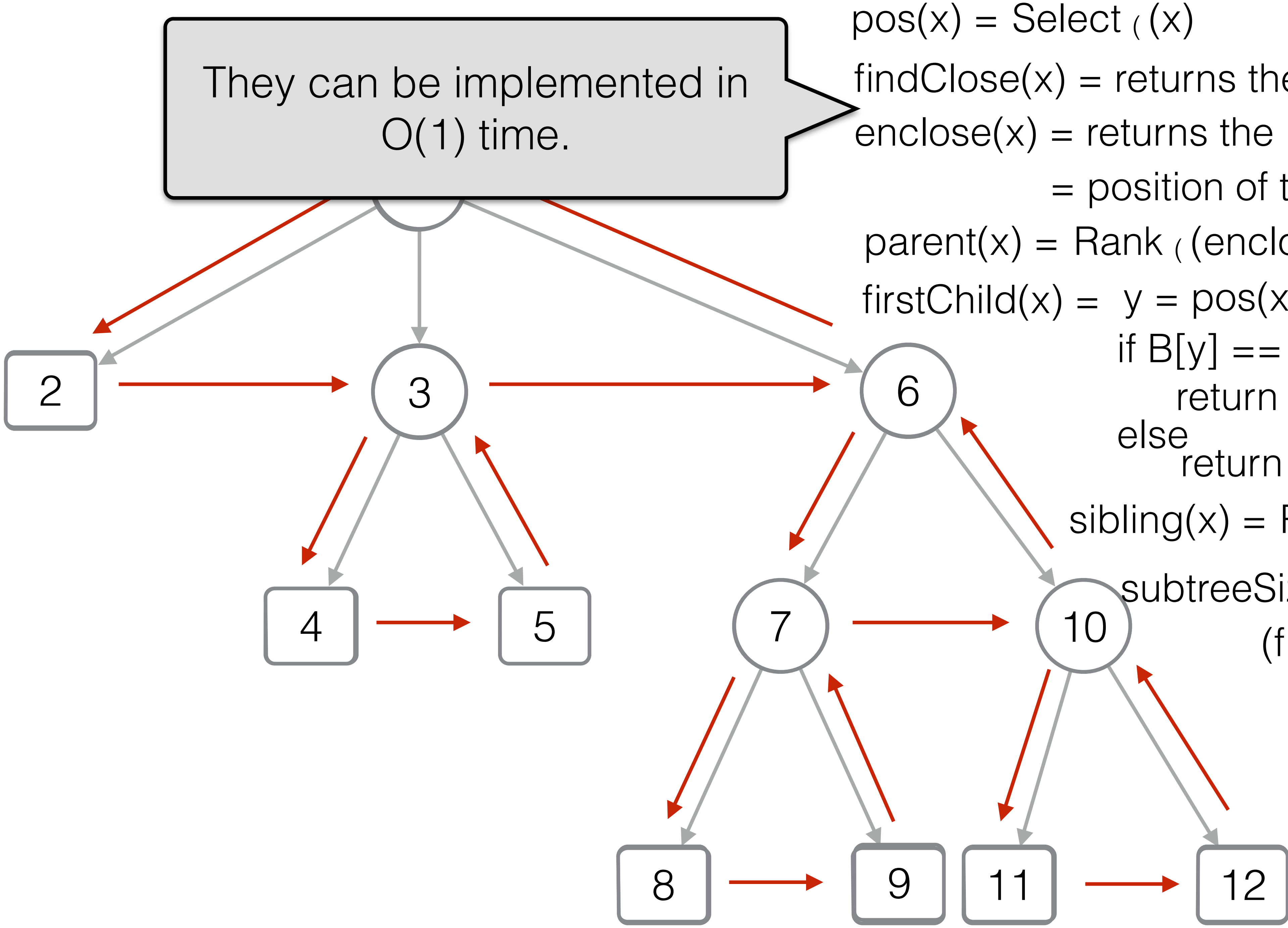
subtreeSize(x) =



B	(	(	)	(	(	)	(	)	)	(	(	(	)	(	)	)	(	(	)	(	)	)	)	)
	1	2	2	3	4	4	5	5	3	6	7	8	8	9	9	7	10	11	11	12	12	10	6	1

Succinct representation of trees (2)

[BP - Balanced parenthesis]



pos(x) = Select $_($ (x)

findClose(x) = returns the position of) matching x-th (

enclose(x) = returns the position of (enclosing x-th (

= position of the parent of x in B

parent(x) = Rank $_($ (enclose(x))

firstChild(x) = y = pos(x)+1

if B[y] ==)

return -1 // is a leaf

else

return Rank $_($ (y)

sibling(x) = Rank $_($ (findClose(x)+1) (if any)

subtreeSize(x) =

(findClose(x) - pos(x) + 1) / 2

B (( ) (( ) ( )) ((( ) ( )) (( ) ( ))))

1 2 2 3 4 4 5 5 3 6 7 8 8 9 9 7 10 11 11 12 12 10 6 1

Succinct representation of trees (2)

[BP - Balanced parenthesis]

> They can be implemented in O(1) time.

pos(x) = Select ((x)

findClose(x) = returns the position of) matching x-th (

enclose(x) = returns the position of (enclosing x-th (

= position of the parent of x in B

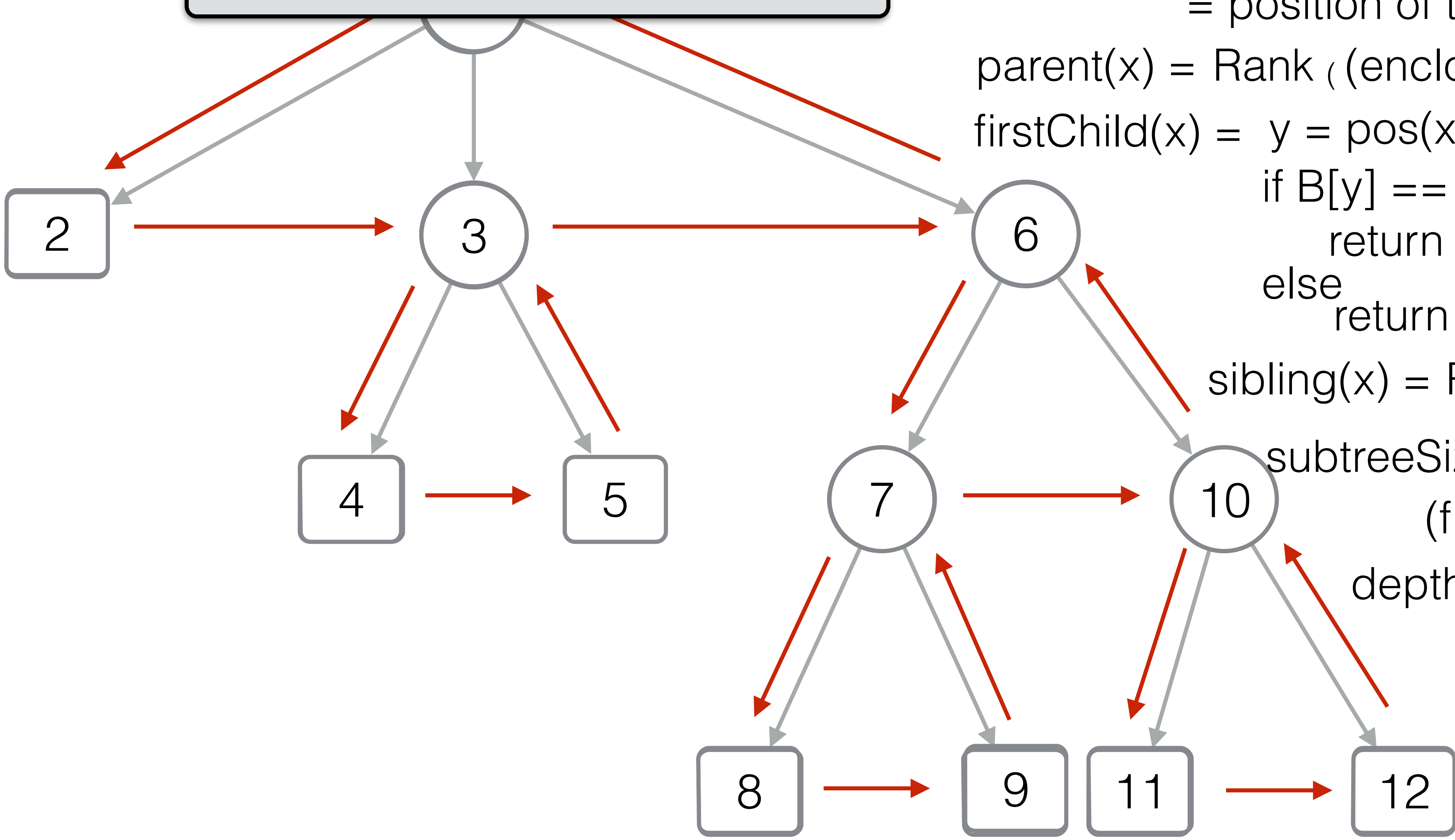
parent(x) = Rank ((enclose(x))

firstChild(x) = y = pos(x)+1
if B[y] ==)
return -1 // is a leaf
else
return Rank ((y)

sibling(x) = Rank ((findClose(x)+1) (if any)

subtreeSize(x) =
(findClose(x) - pos(x) + 1) / 2

depth(x) =



B (( ) (( ) ( )) ((( ) ( )) (( ) ( ))))

1 2 2 3 4 4 5 5 3 6 7 8 8 9 9 7 10 11 11 12 12 10 6 1

Succinct representation of trees (2)

[BP - Balanced parenthesis]

pos(x) = Select$_($ (x)

findClose(x) = returns the position of) matching x-th (

enclose(x) = returns the position of (enclosing x-th (

= position of the parent of x in B

parent(x) = Rank$_($ (enclose(x))

firstChild(x) = y = pos(x)+1

if B[y] ==)

return -1 // is a leaf

else

return Rank$_($ (y)

sibling(x) = Rank$_($ (findClose(x)+1) (if any)

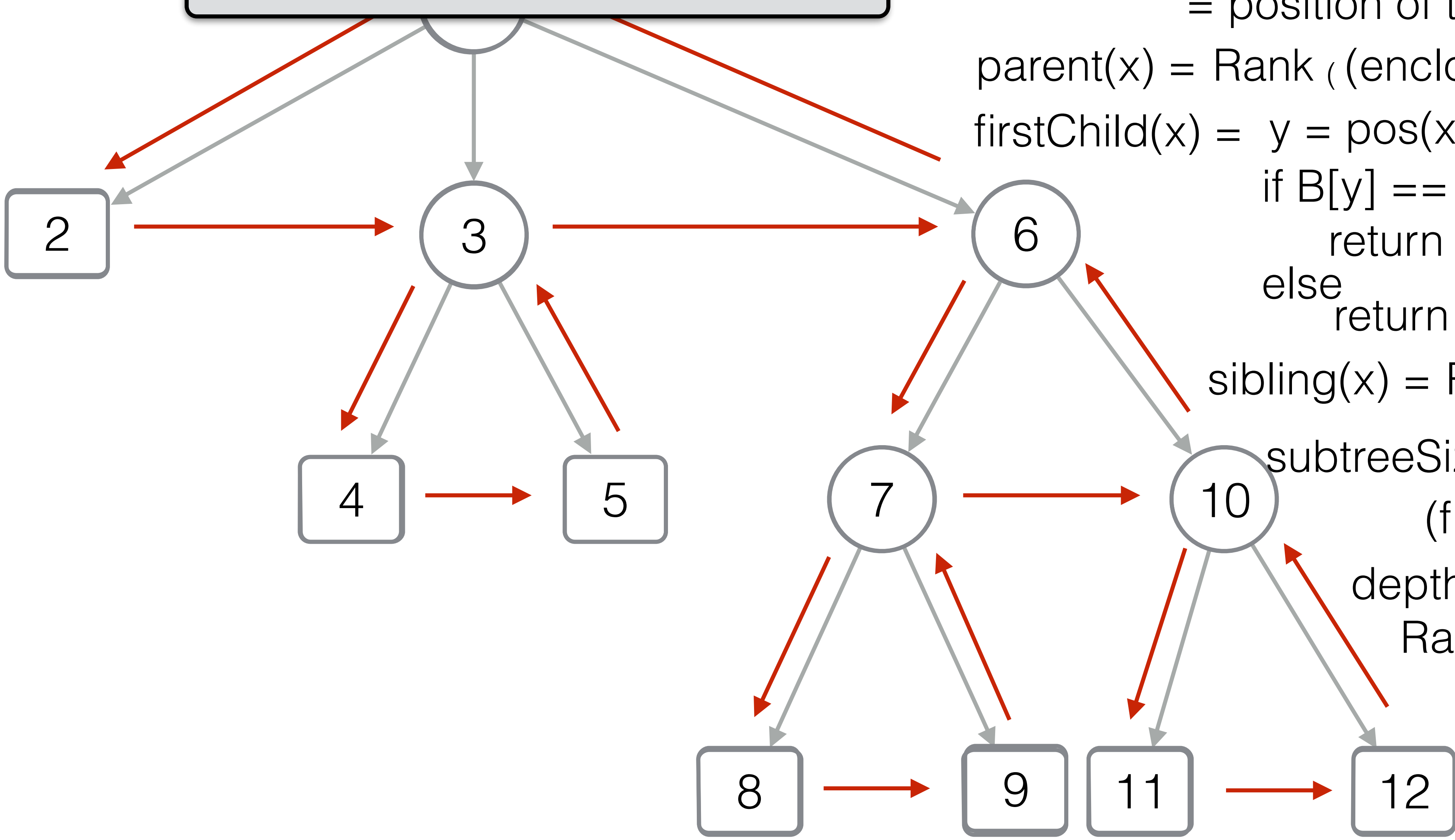
subtreeSize(x) =

(findClose(x) - pos(x) + 1) / 2

depth(x) =

Rank$_($ (pos(x)) - Rank$_)$ (pos(x))

They can be implemented in O(1) time.



B (( ) (( ) ( )) ((( ) ( )) (( ) ( ))))

1 2 2 3 4 4 5 5 3 6 7 8 8 9 9 7 10 11 11 12 12 10 6 1

Succinct representation of trees (2)

[BP - Balanced parenthesis]

They can be implemented in O(1) time.

$pos(x) = Select_{(}(x)$

findClose(x) = returns the position of) matching x-th (

enclose(x) = returns the position of (enclosing x-th (

= position of the parent of x in B

$parent(x) = Rank_{(}(enclose(x))$

firstChild(x) = y = pos(x)+1

if B[y] ==)

return -1 // is a leaf

else

$return\ Rank_{(}(y)$

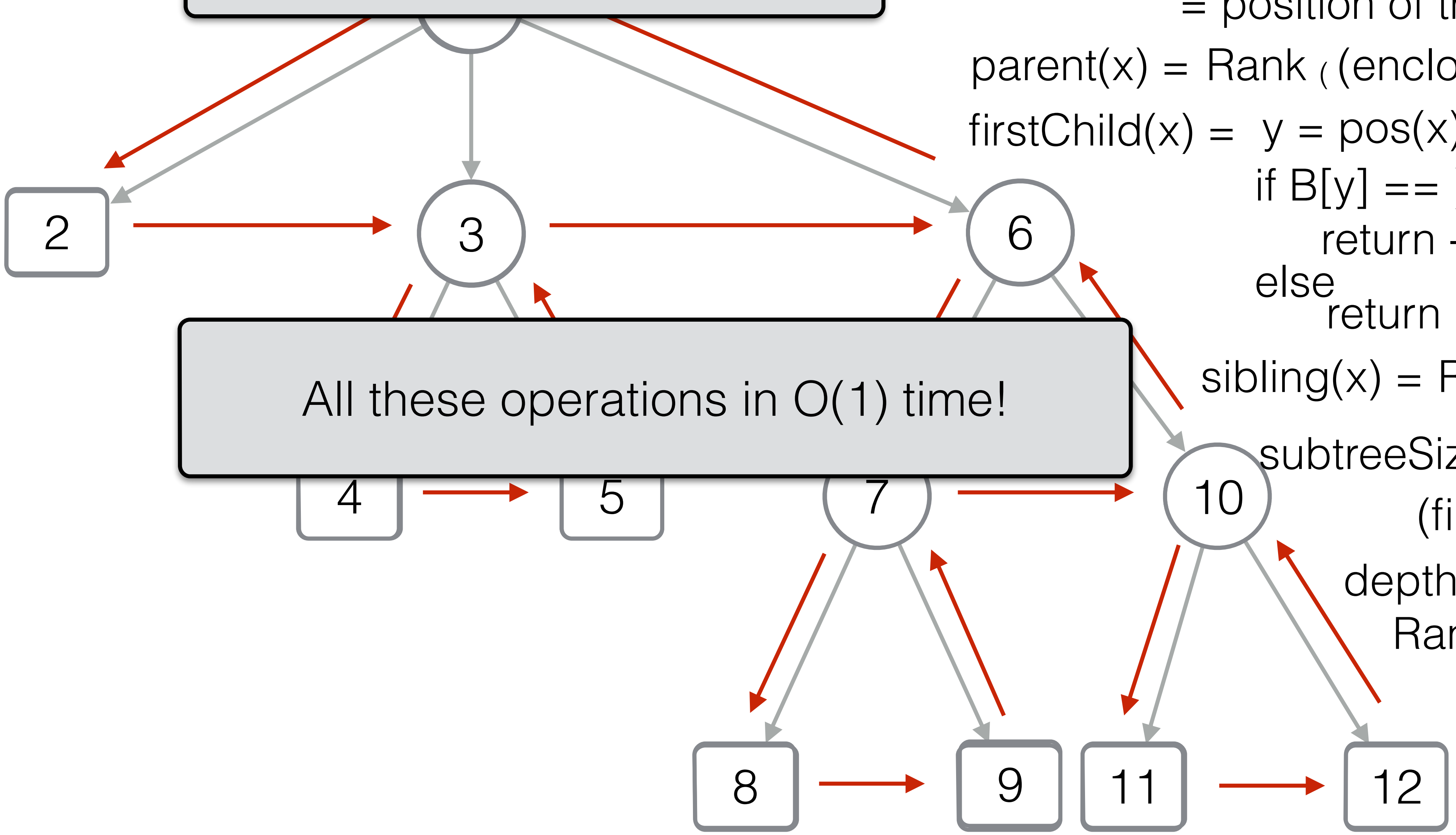
$sibling(x) = Rank_{(}(findClose(x)+1)$ (if any)

subtreeSize(x) =

(findClose(x) - pos(x) + 1) / 2

depth(x) =

$Rank_{(}(pos(x)) - Rank_{)}(pos(x))$



All these operations in O(1) time!

B (( ) (( ) ( )) ((( ) ( )) (( ) ( ))))

1 2 2 3 4 4 5 5 3 6 7 8 8 9 9 7 10 11 11 12 12 10 6 1

Succinct representation of trees (2)

[BP - Balanced parenthesis]

They can be implemented in O(1) time.

pos(x) = Select$_{(}$(x)

findClose(x) = returns the position of) matching x-th (

enclose(x) = returns the position of (enclosing x-th (

= position of the parent of x in B

parent(x) = Rank$_{(}$(enclose(x))

firstChild(x) = y = pos(x)+1

if B[y] ==)

return -1 // is a leaf

else

return Rank$_{(}$(y)

sibling(x) = Rank$_{(}$(findClose(x)+1) (if any)

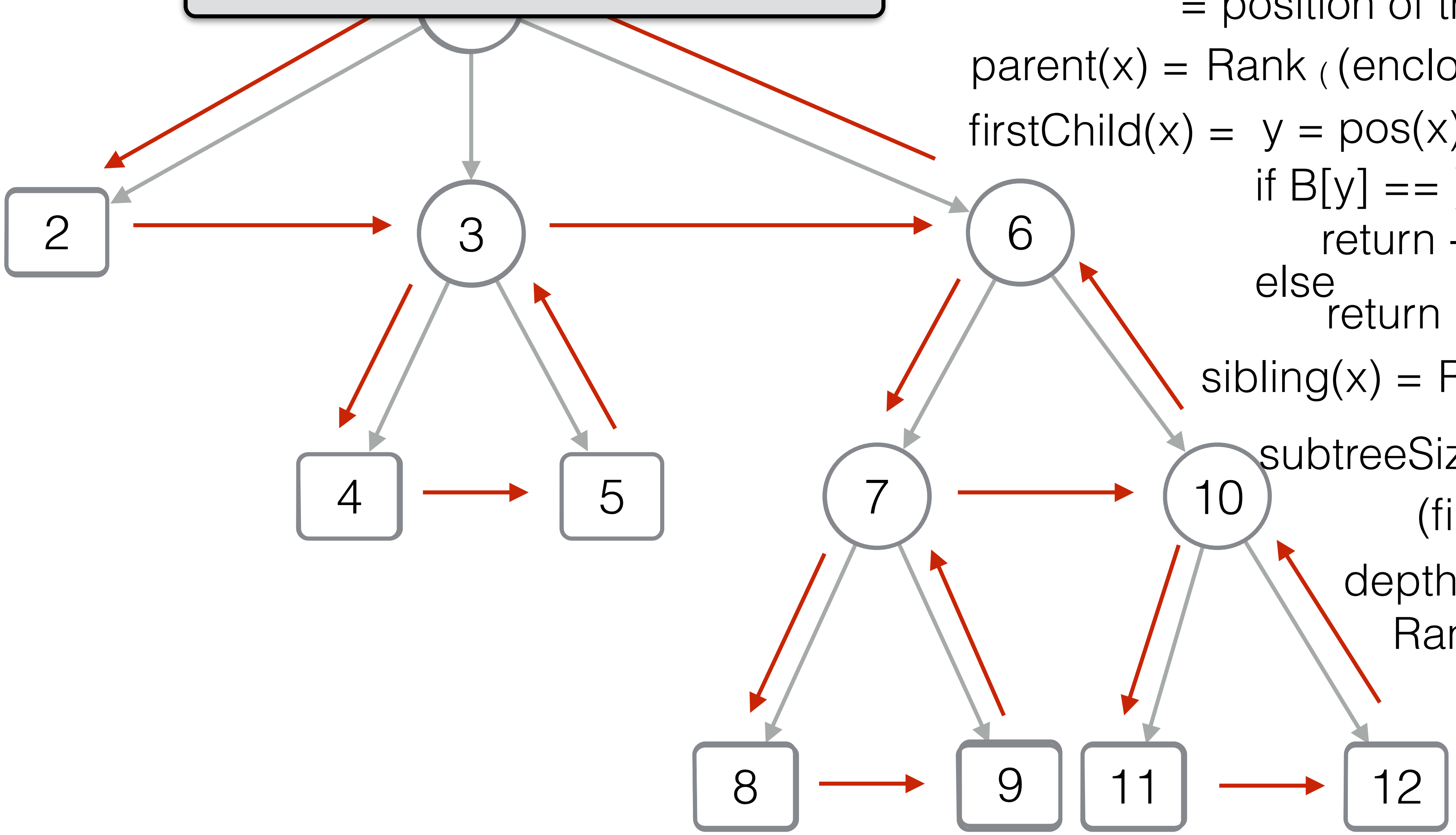
subtreeSize(x) =

(findClose(x) - pos(x) + 1) / 2

depth(x) =

Rank$_{(}$(pos(x)) - Rank$_{)}$(pos(x))

degree(x) = ?



B	(	(	)	(	(	)	(	)	)	(	(	(	)	(	)	)	(	(	)	(	)	)	)	)
	1	2	2	3	4	4	5	5	3	6	7	8	8	9	9	7	10	11	11	12	12	10	6	1

Succinct representation of trees (2)

[BP - Balanced parenthesis]

They can be implemented in O(1) time.

pos(x) = Select$_{(}$ (x)

findClose(x) = returns the position of) matching x-th (

enclose(x) = returns the position of (enclosing x-th (

= position of the parent of x in B

parent(x) = Rank$_{(}$ (enclose(x))

firstChild(x) = y = pos(x)+1

if B[y] ==)

return -1 // is a leaf

else

return Rank$_{(}$ (y)

sibling(x) = Rank$_{(}$ (findClose(x)+1) (if any)

subtreeSize(x) =

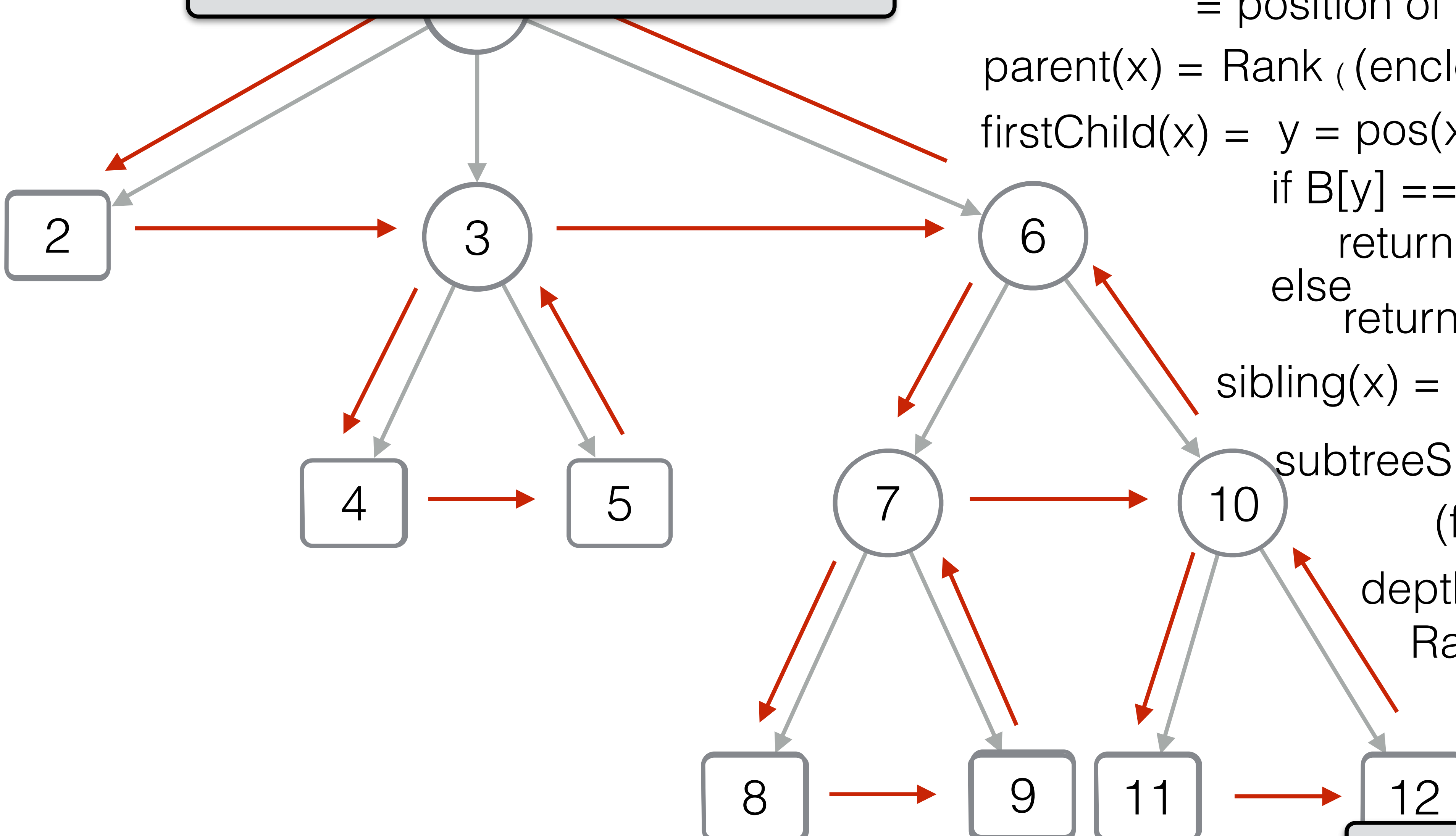
(findClose(x) - pos(x) + 1) / 2

depth(x) =

Rank$_{(}$ (pos(x)) - Rank$_{)}$ (pos(x))

degree(x) = ?

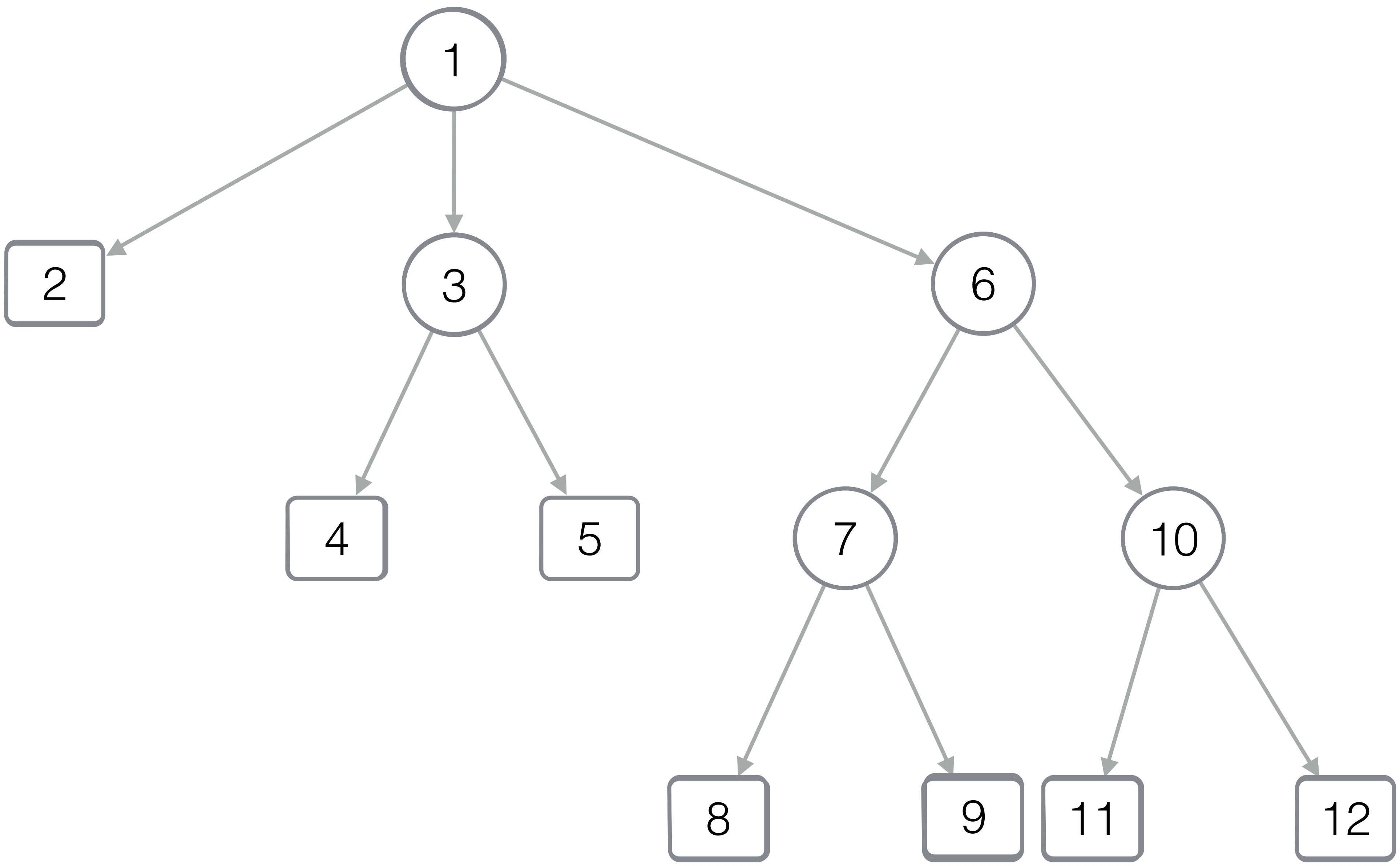
Quite inefficient!
Solved by repeatedly calling sibling to scan x's children.



B	(	(	)	(	(	)	(	)	)	(	(	(	)	(	)	)	(	(	)	(	)	)	)	)
	1	2	2	3	4	4	5	5	3	6	7	8	8	9	9	7	10	11	11	12	12	10	6	1

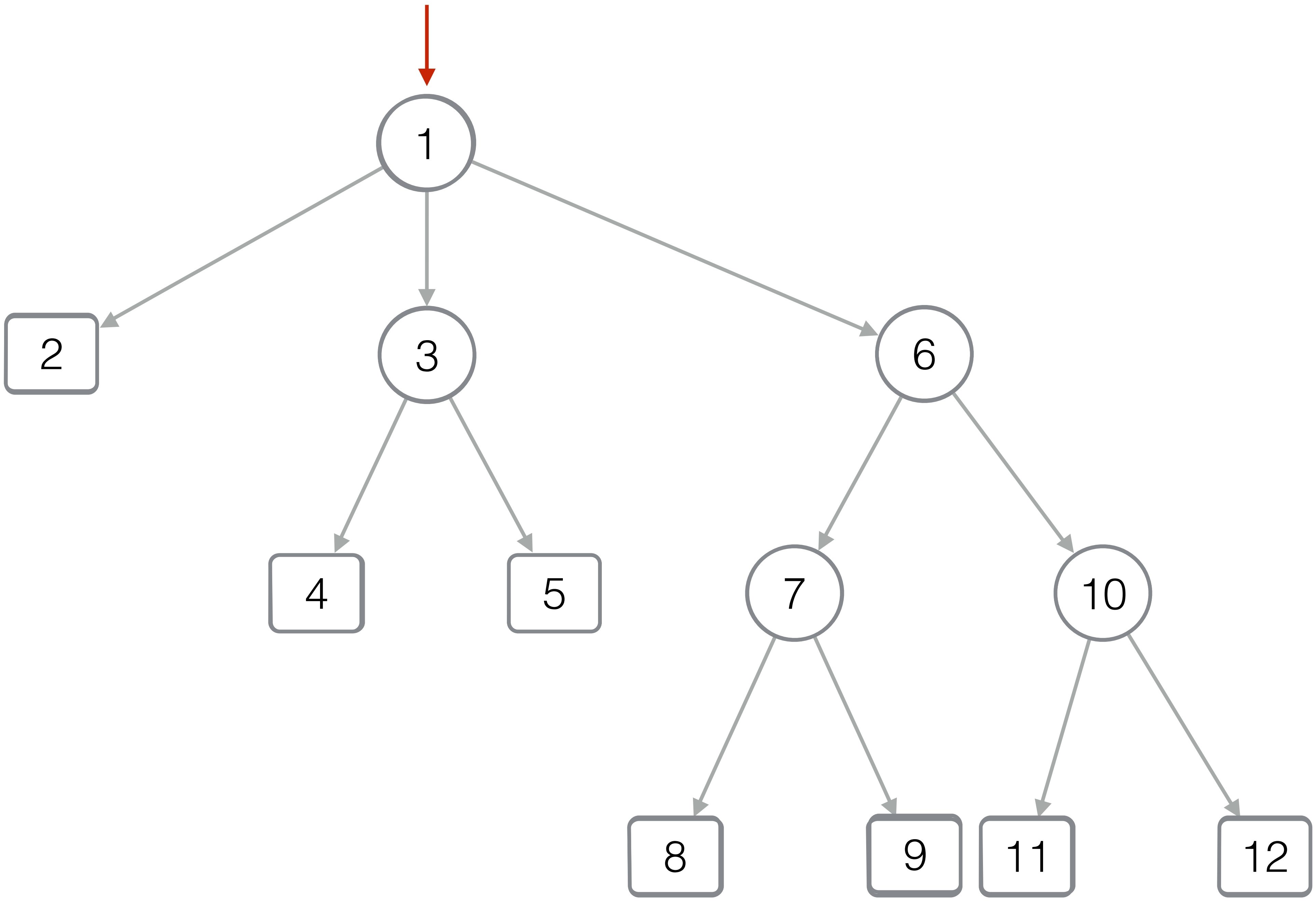
Succinct representation of trees (3)

[DFUDS - Depth First Unary Degree Sequence



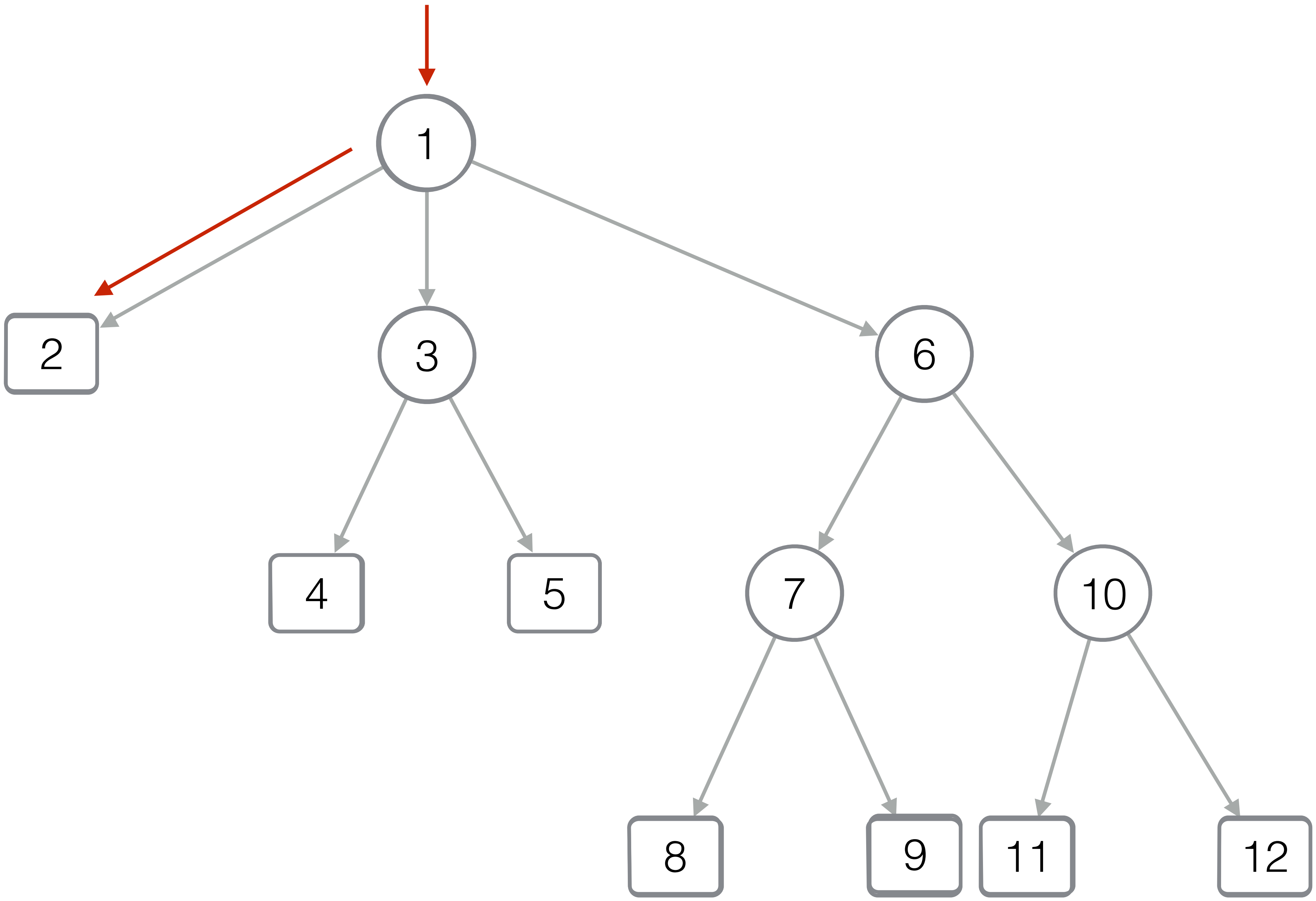
Succinct representation of trees (3)

[DFUDS - Depth First Unary Degree Sequence



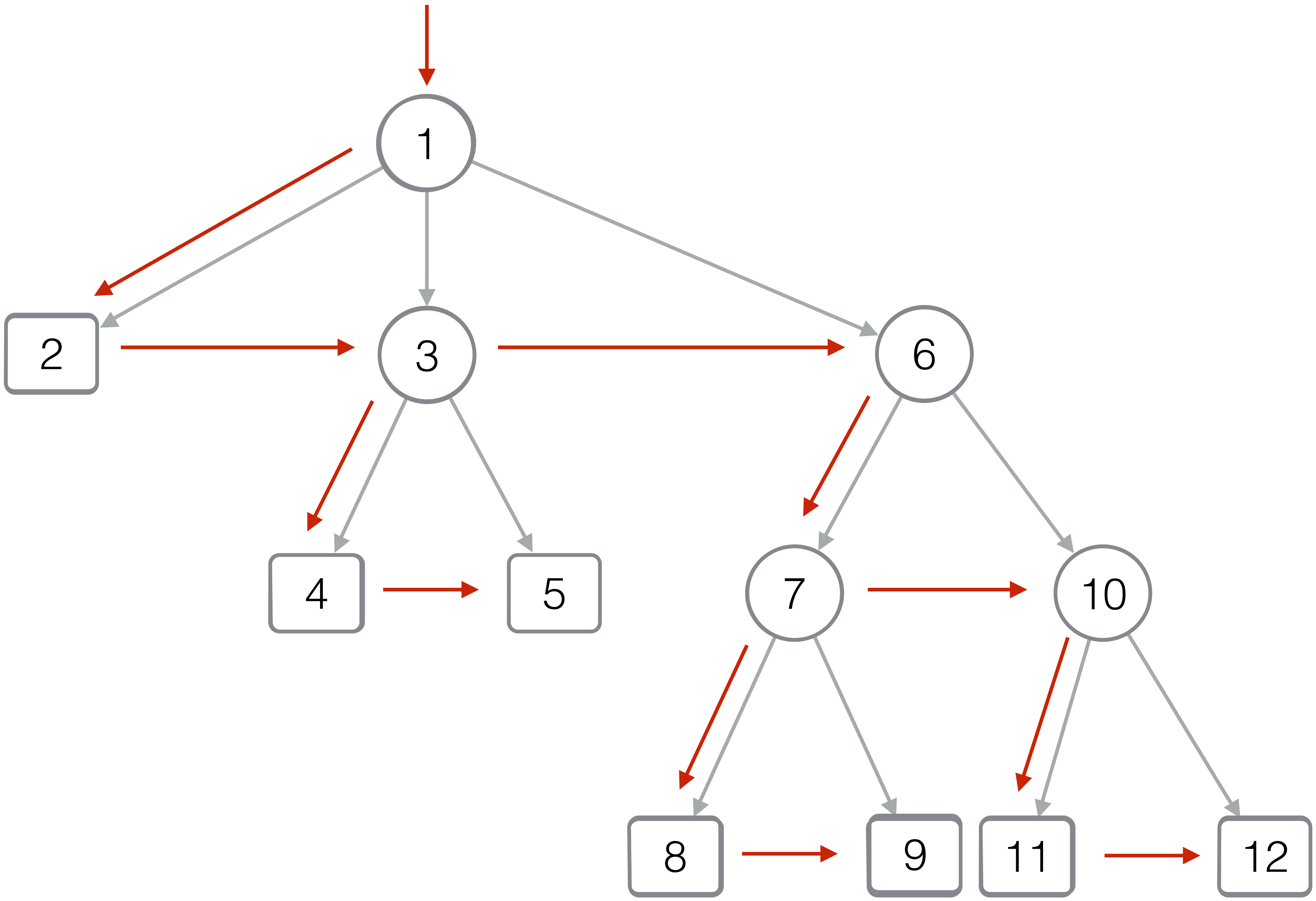
Succinct representation of trees (3)

[DFUDS - Depth First Unary Degree Sequence



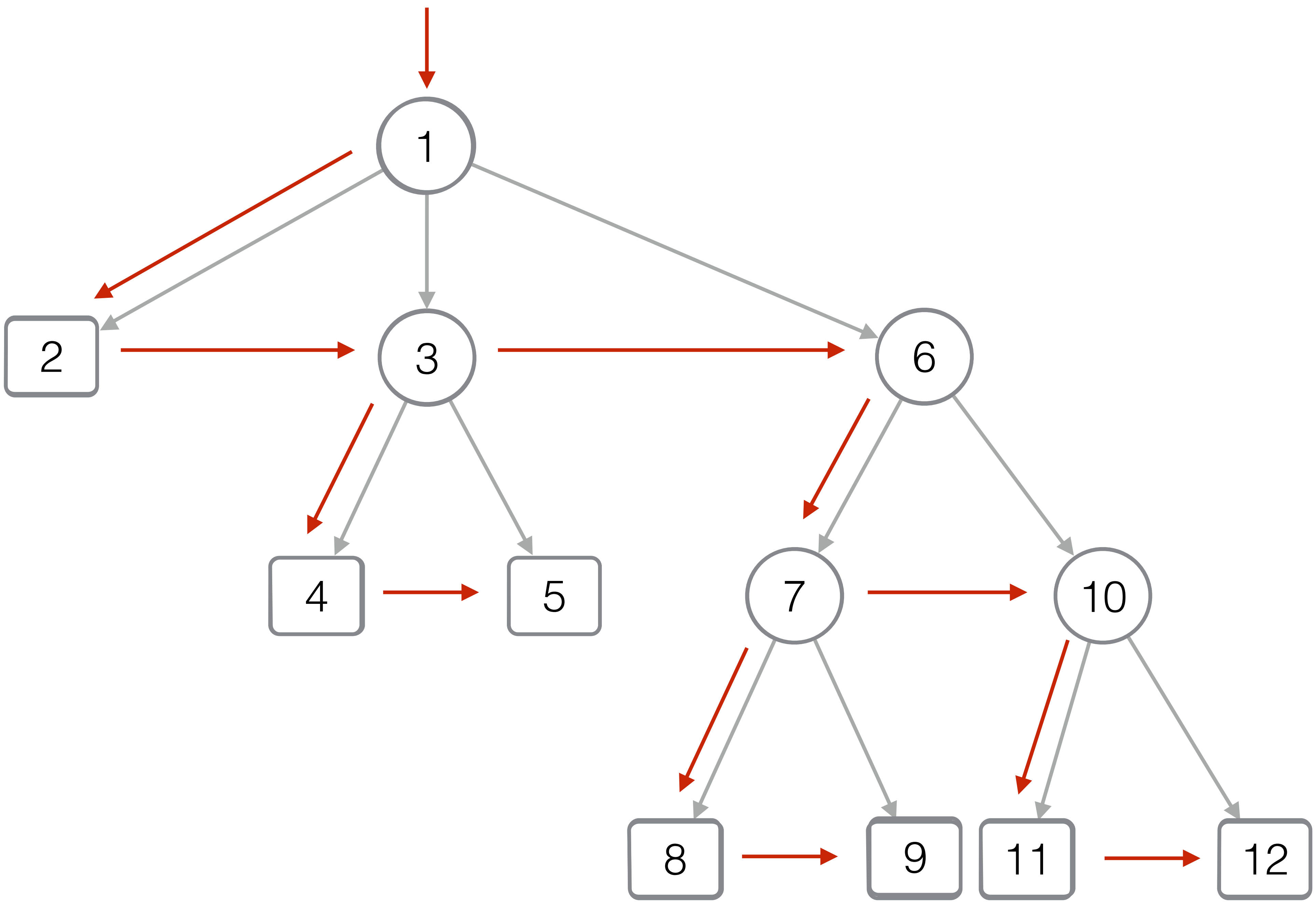
Succinct representation of trees (3)

[DFUDS - Depth First Unary Degree Sequence



Succinct representation of trees (3)

[DFUDS - Depth First Unary Degree Sequence

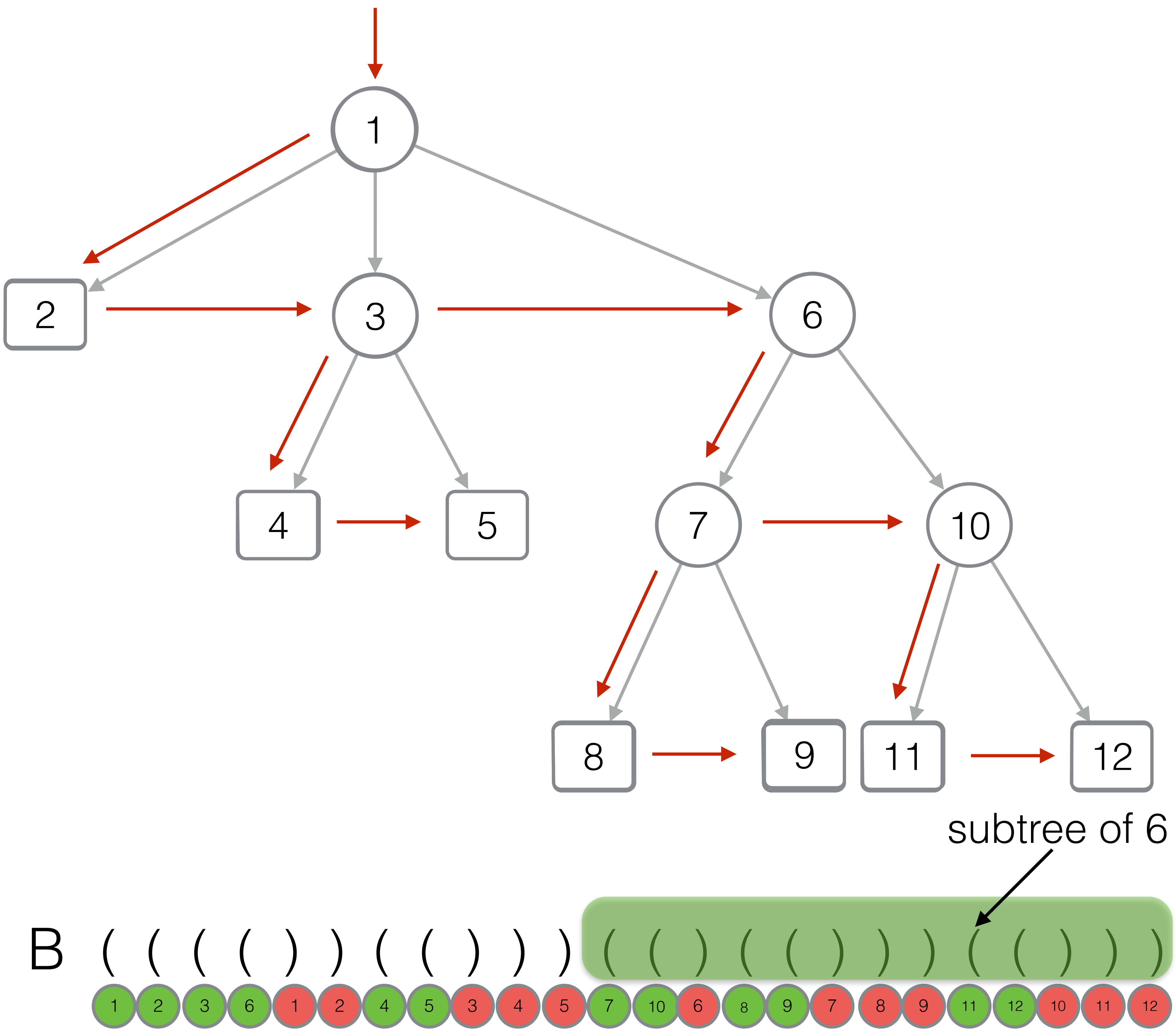


B (((( )) (( ))) (( ) (( ))) (( )))

1 2 3 6 1 2 4 5 3 4 5 7 10 6 8 9 7 8 9 11 12 10 11 12

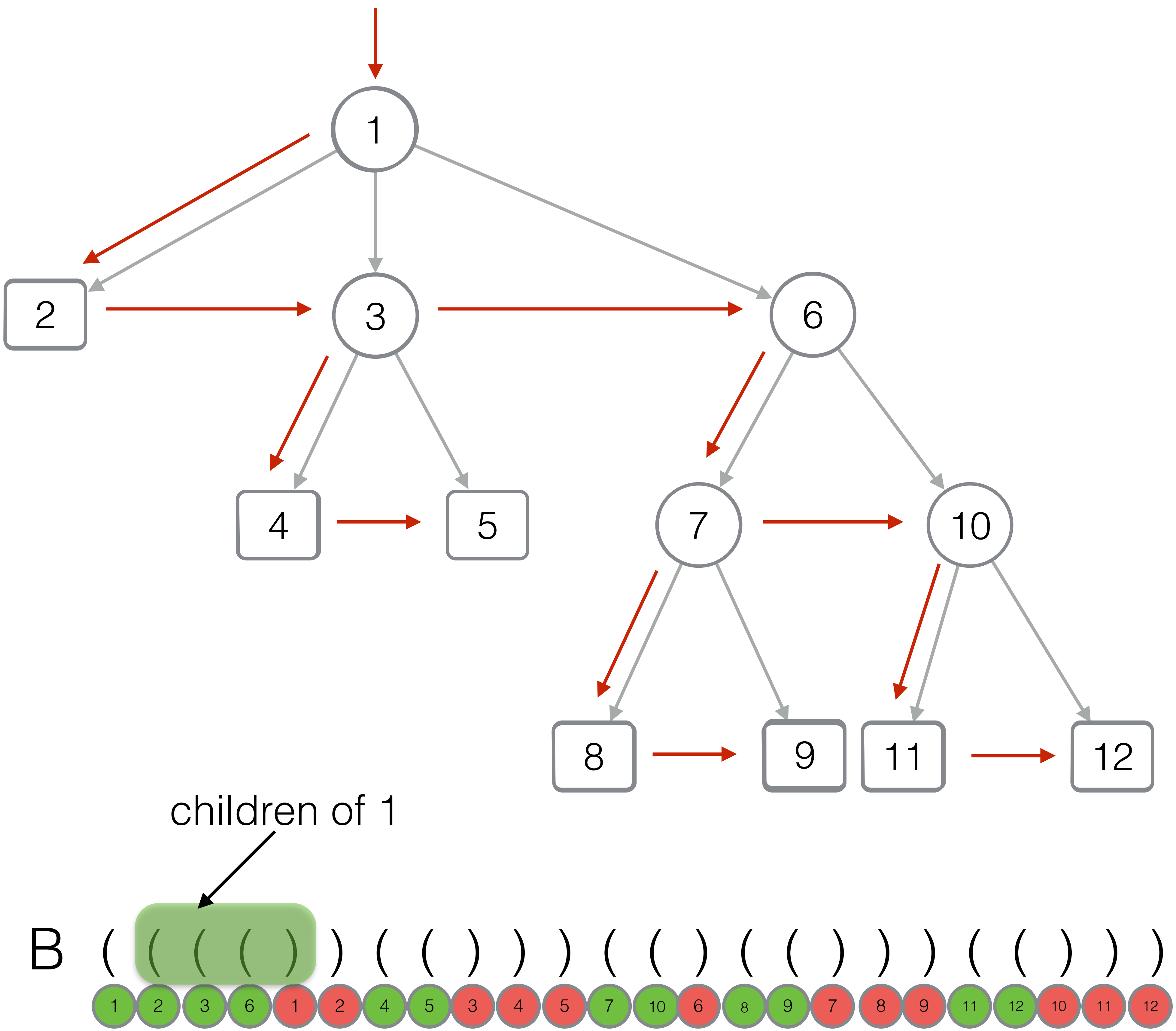
Succinct representation of trees (3)

[DFUDS - Depth First Unary Degree Sequence



Succinct representation of trees (3)

[DFUDS - Depth First Unary Degree Sequence



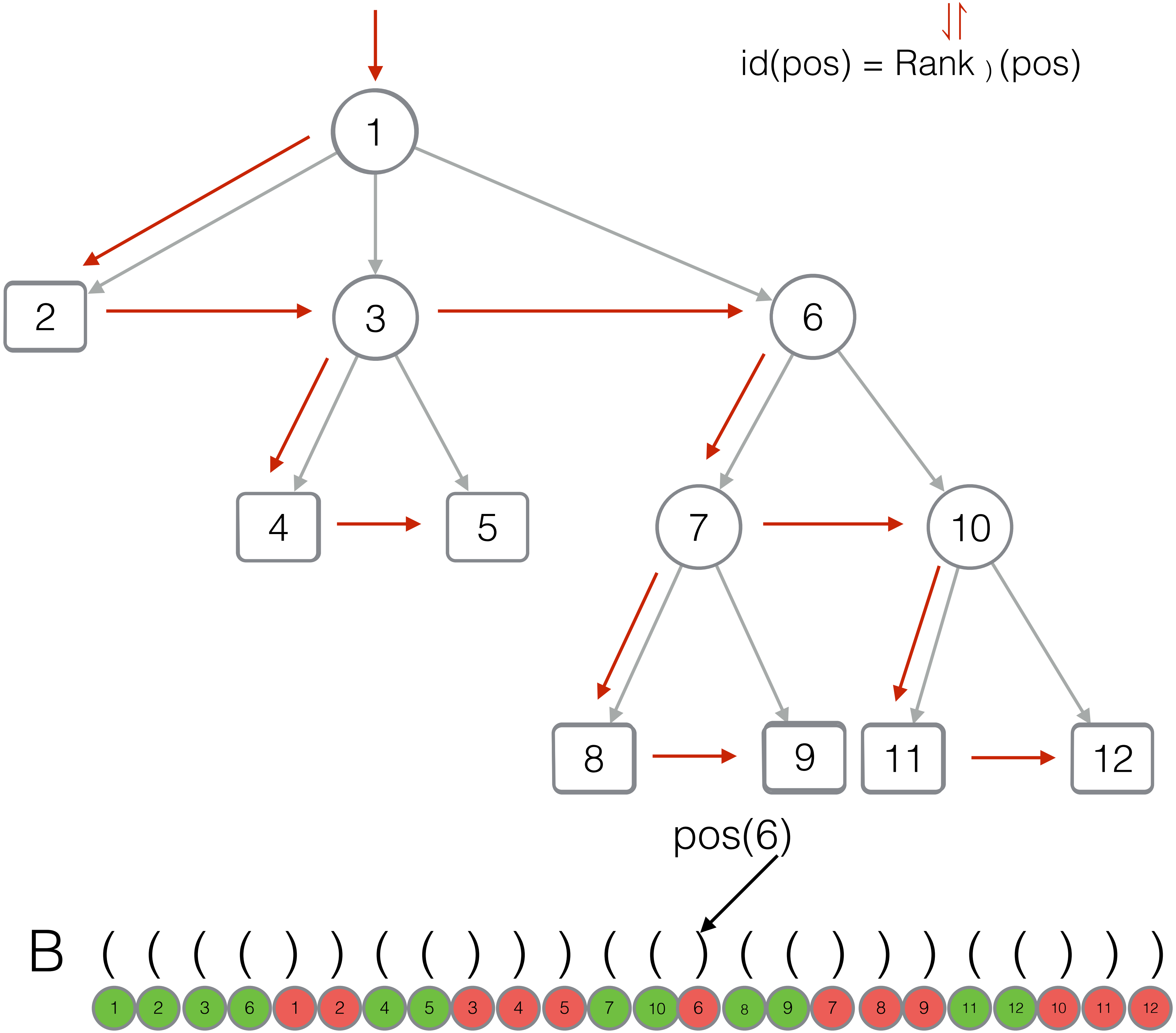
Succinct representation of trees (3)

[DFUDS - Depth First Unary Degree Sequence

$pos(x) = Select_{)}(x)$ // closing)

⇃↾

$id(pos) = Rank_{)}(pos)$



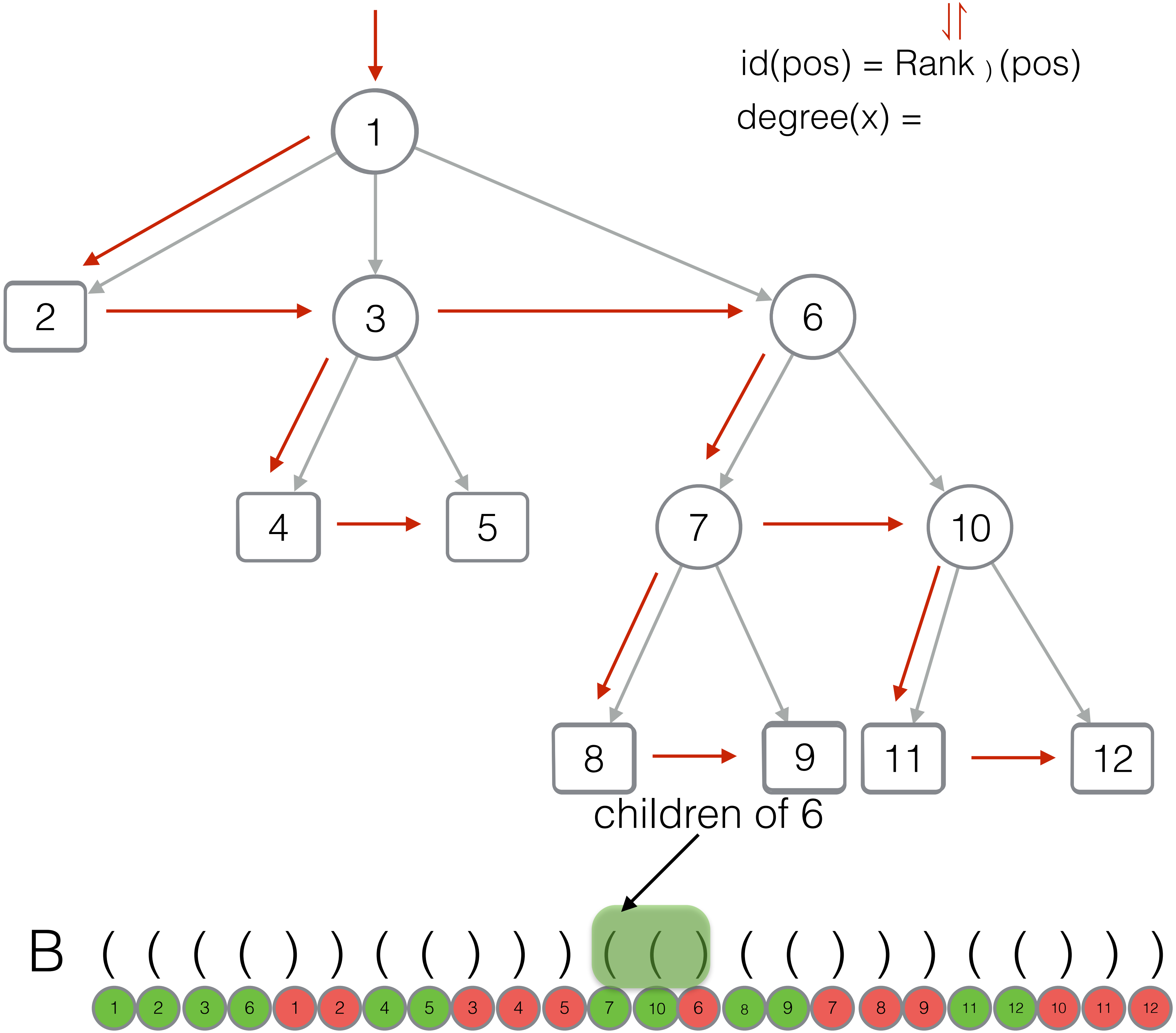
Succinct representation of trees (3)

[DFUDS - Depth First Unary Degree Sequence

pos(x) = $Select_{)}(x)$ // closing)

id(pos) = $Rank_{)}(pos)$

degree(x) =



Succinct representation of trees (3)

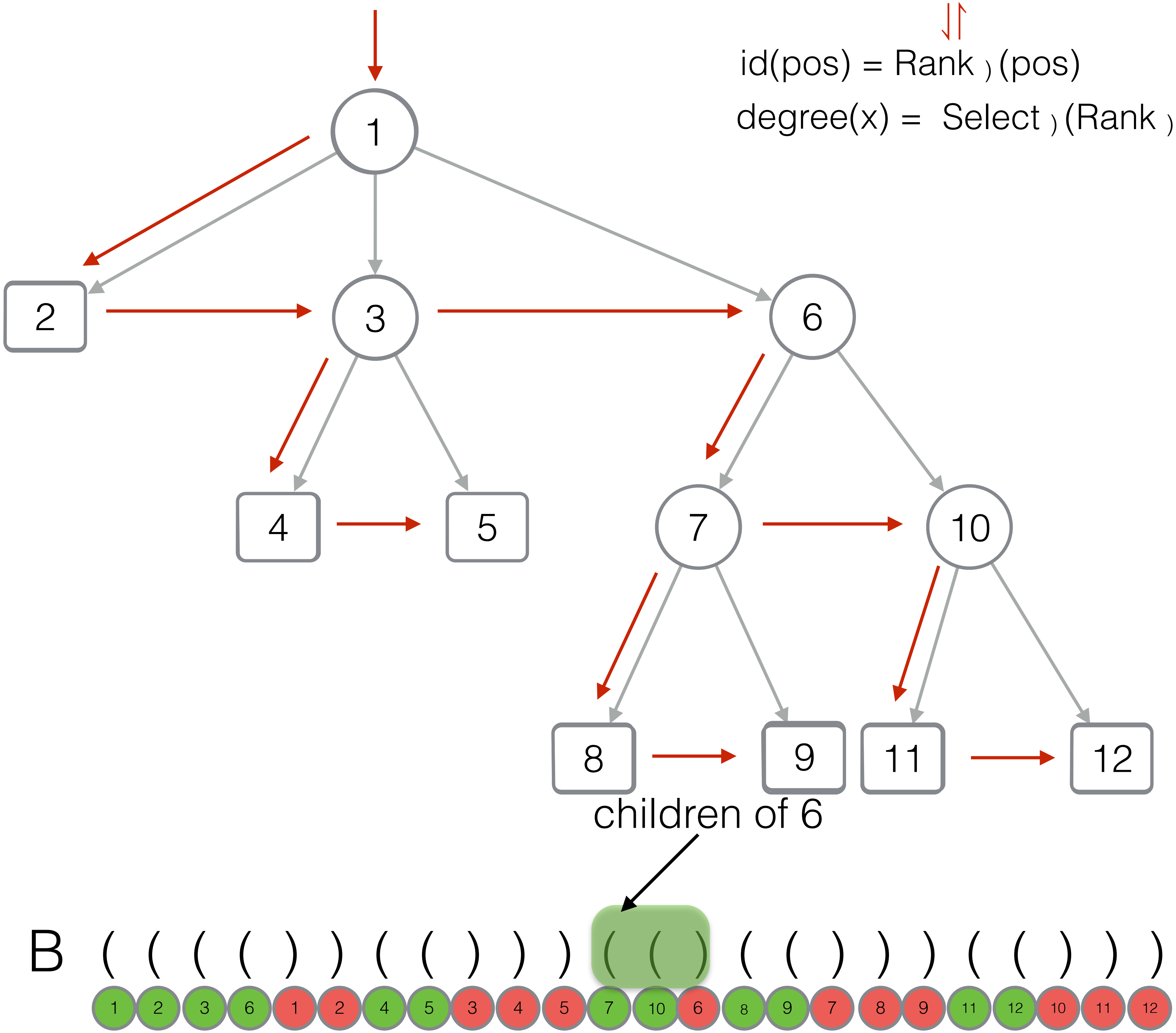
[DFUDS - Depth First Unary Degree Sequence

pos(x) = $Select_{)}(x)$ // closing)

⇃↾

id(pos) = $Rank_{)}(pos)$

degree(x) = $Select_{)}(Rank_{)}(pos(x))) - x$



Succinct representation of trees (3)

[DFUDS - Depth First Unary Degree Sequence

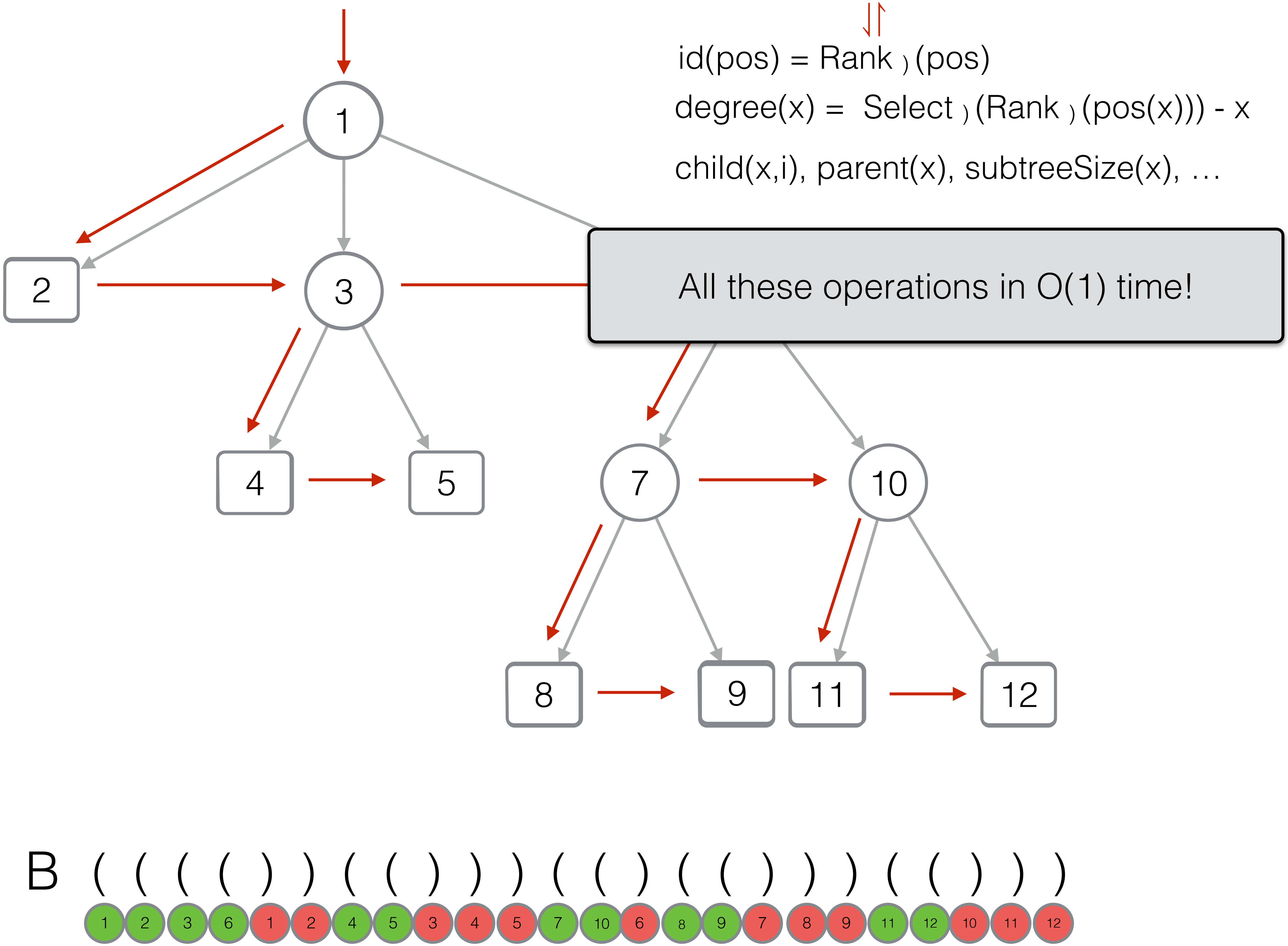
$pos(x) = \mathrm{Select}_{)}(x)$ // closing)

⇃↾

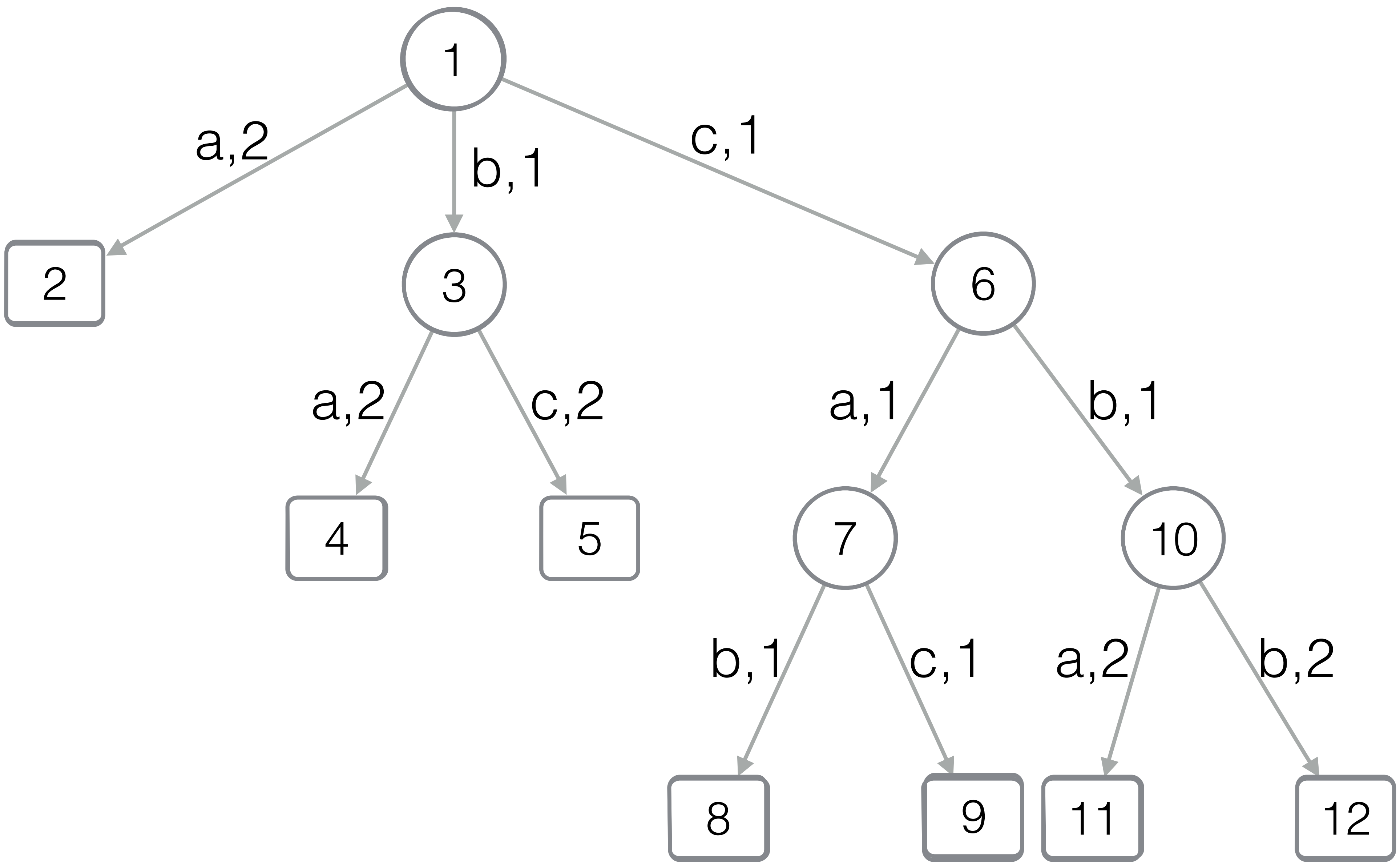
$id(pos) = \mathrm{Rank}_{)}(pos)$

$degree(x) = \mathrm{Select}_{)}(\mathrm{Rank}_{)}(pos(x))) - x$

child(x,i), parent(x), subtreeSize(x), …



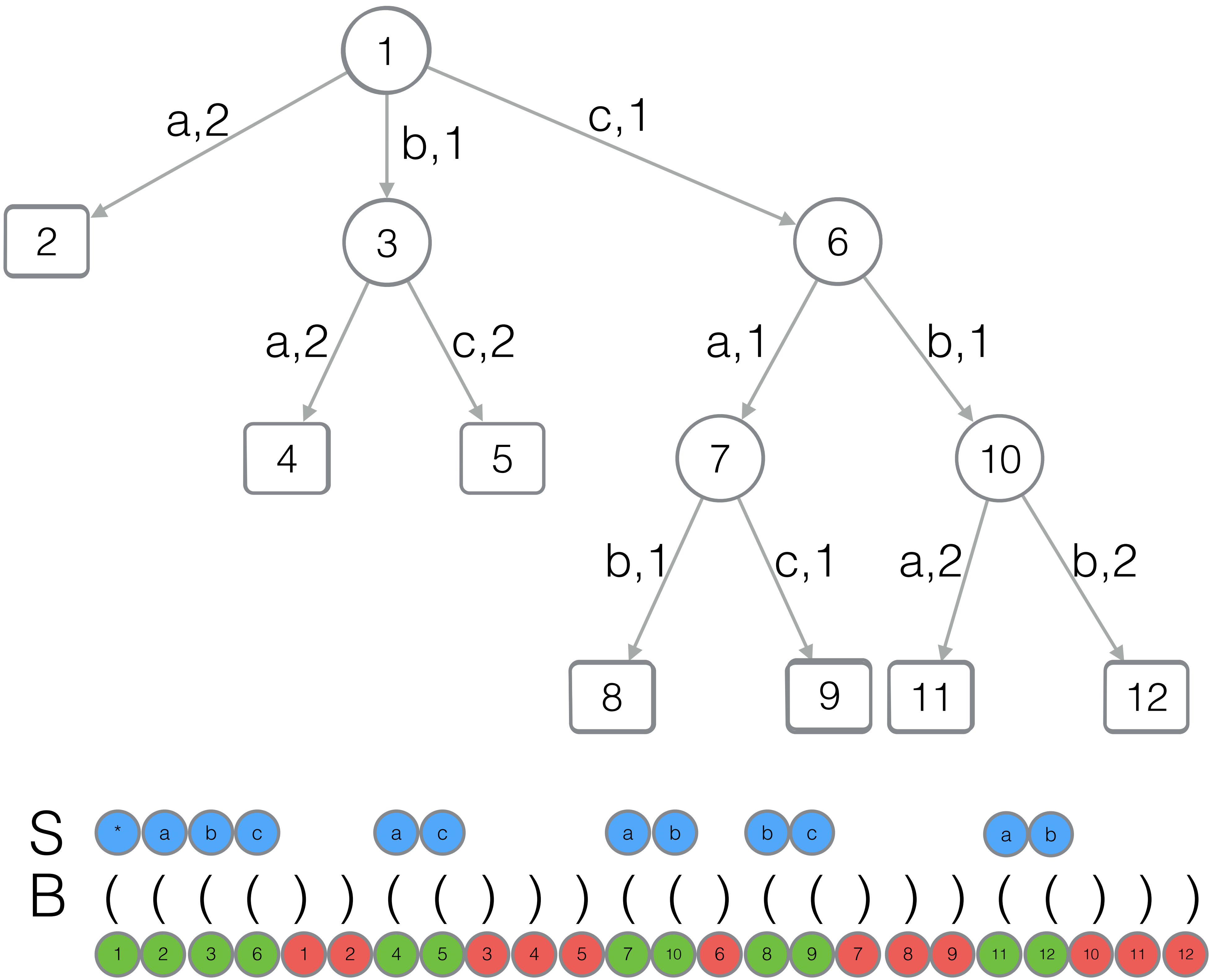
Patricia trie with DFUDS



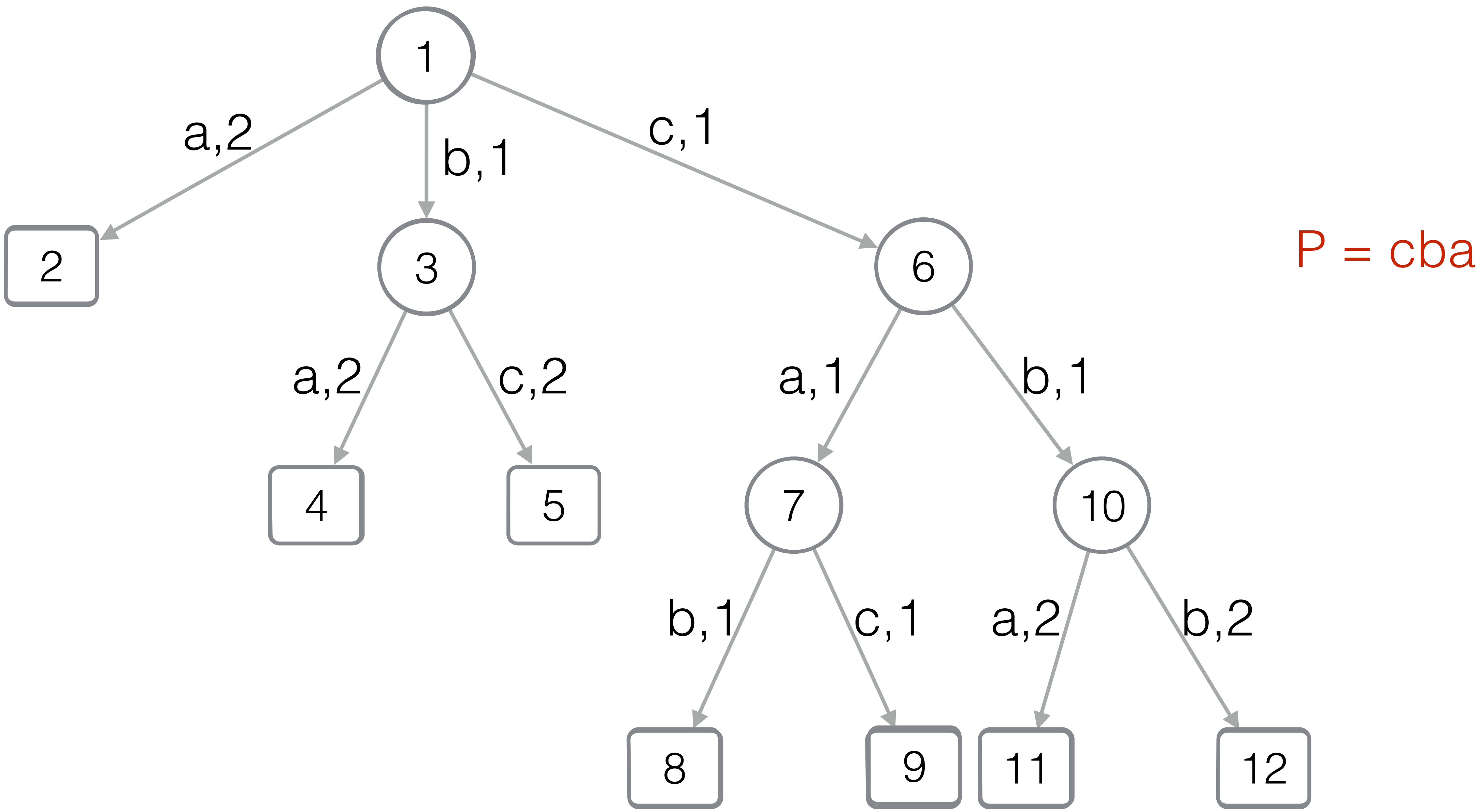
B (((( )) (( ))) (( ) (( ))) (( )))

1 2 3 6 1 2 4 5 3 4 5 7 10 6 8 9 7 8 9 11 12 10 11 12

Patricia trie with DFUDS



Patricia trie with DFUDS

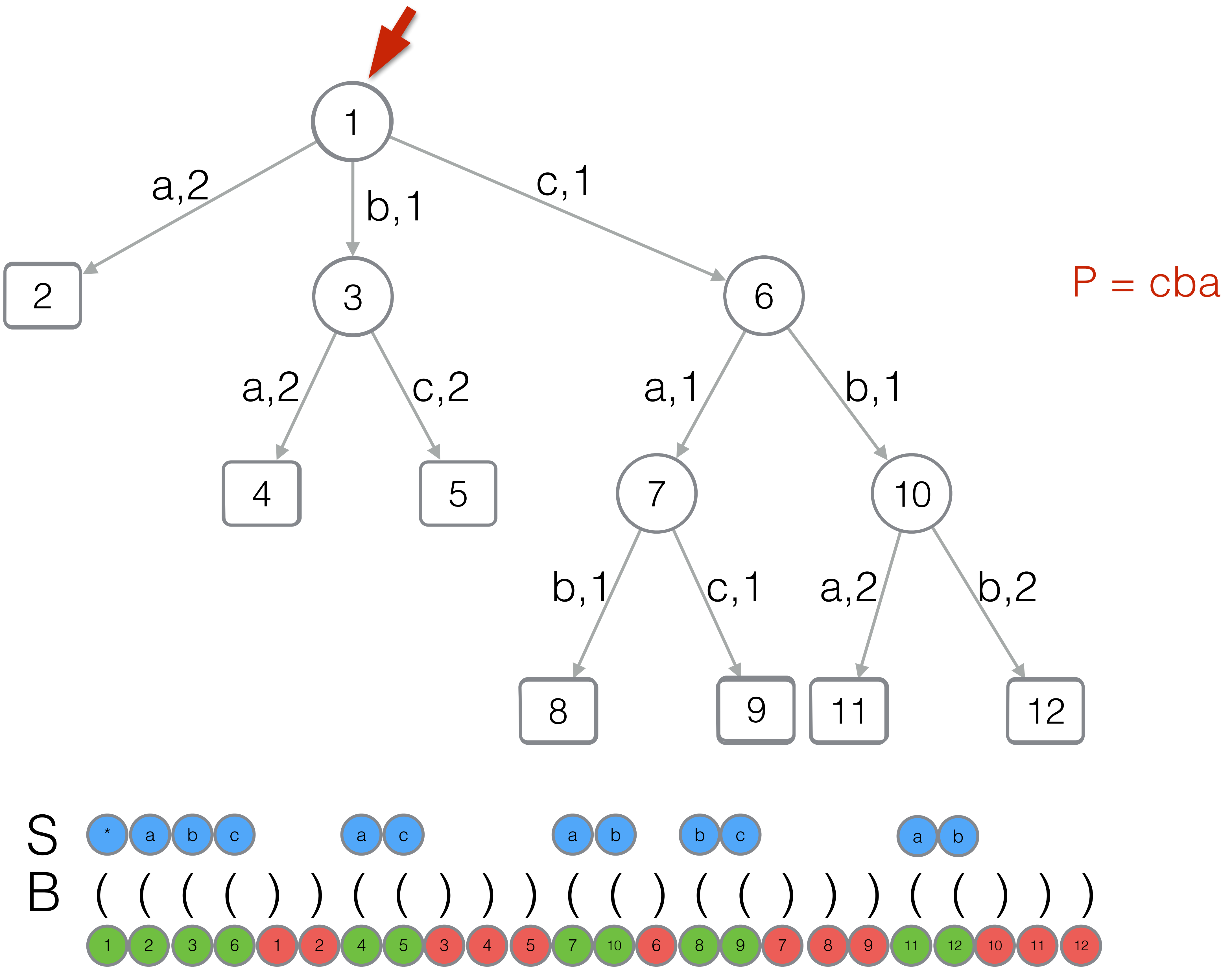


S: * a b c   a c   a b   b c   a b

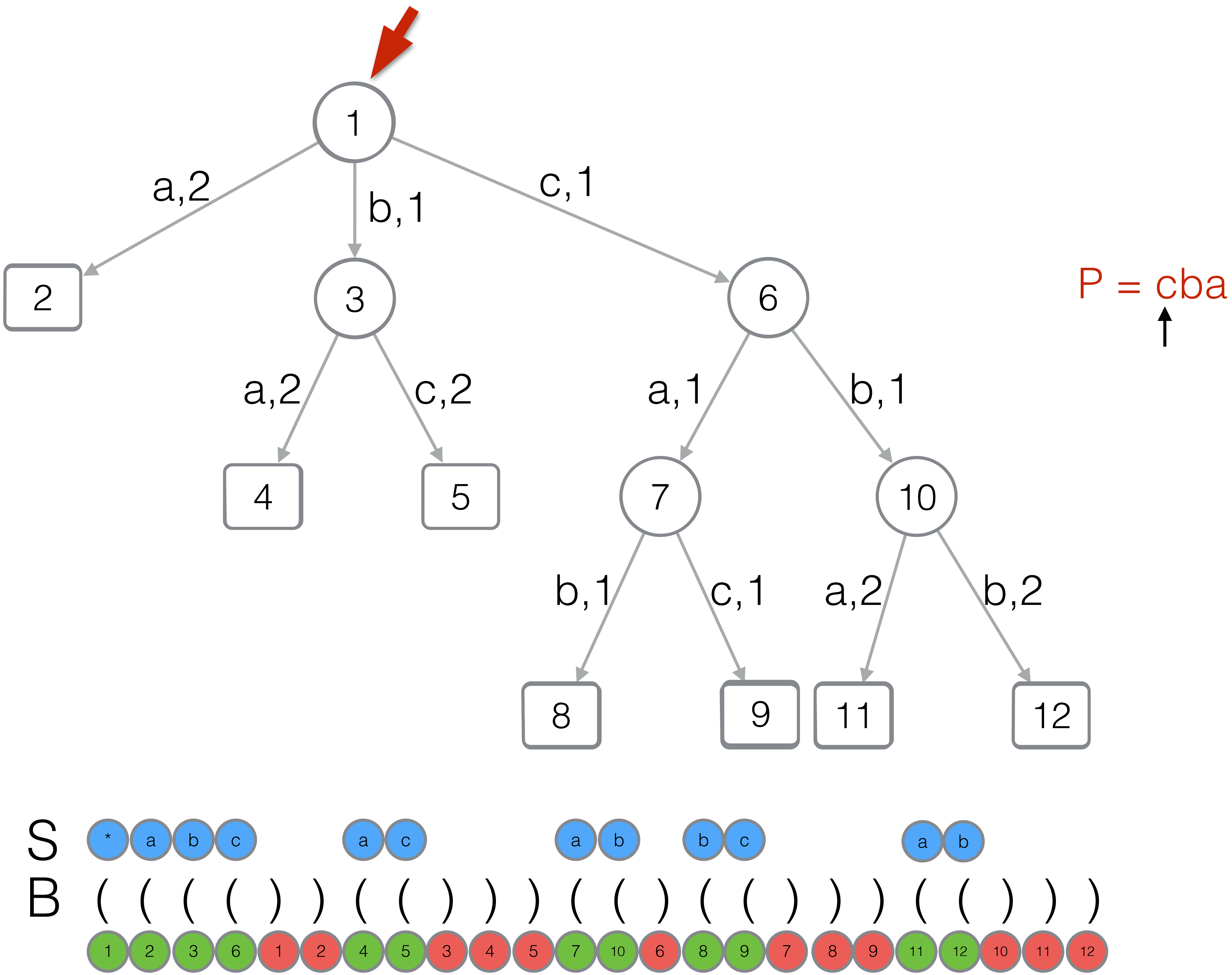
B: (((( )) (( ))) (( ) (( ))) (( )))

1 2 3 6 1 2 4 5 3 4 5 7 10 6 8 9 7 8 9 11 12 10 11 12

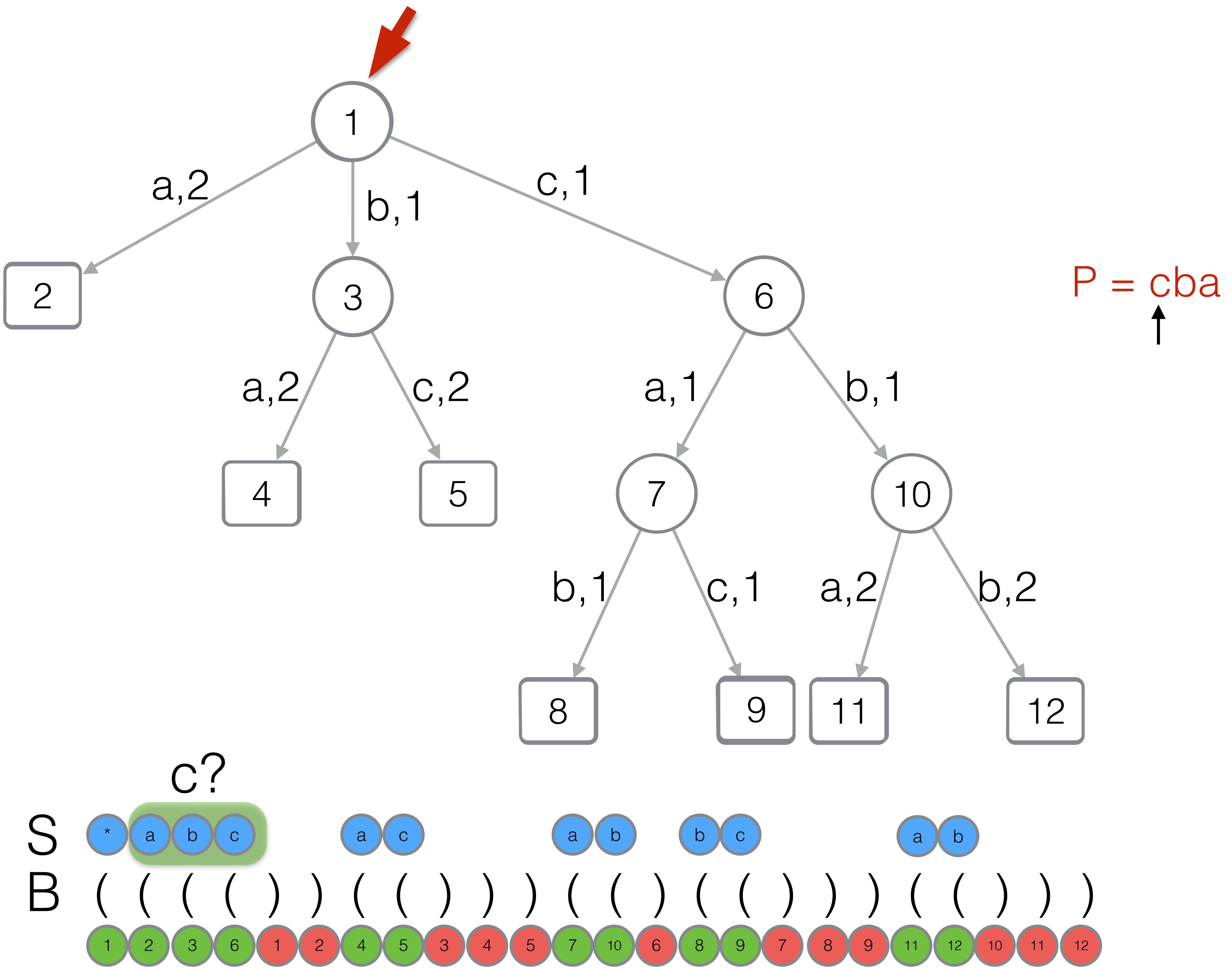
Patricia trie with DFUDS



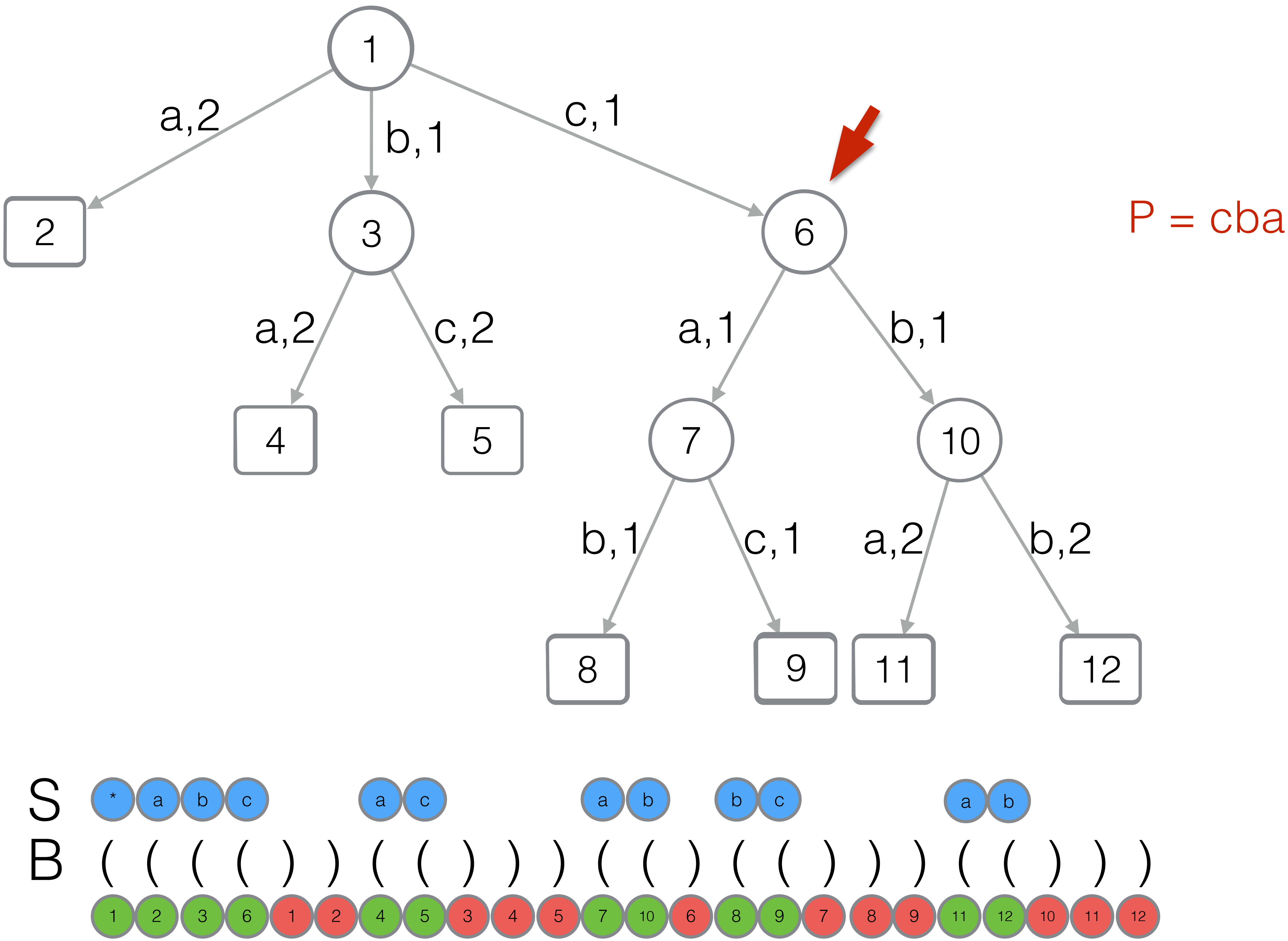
Patricia trie with DFUDS



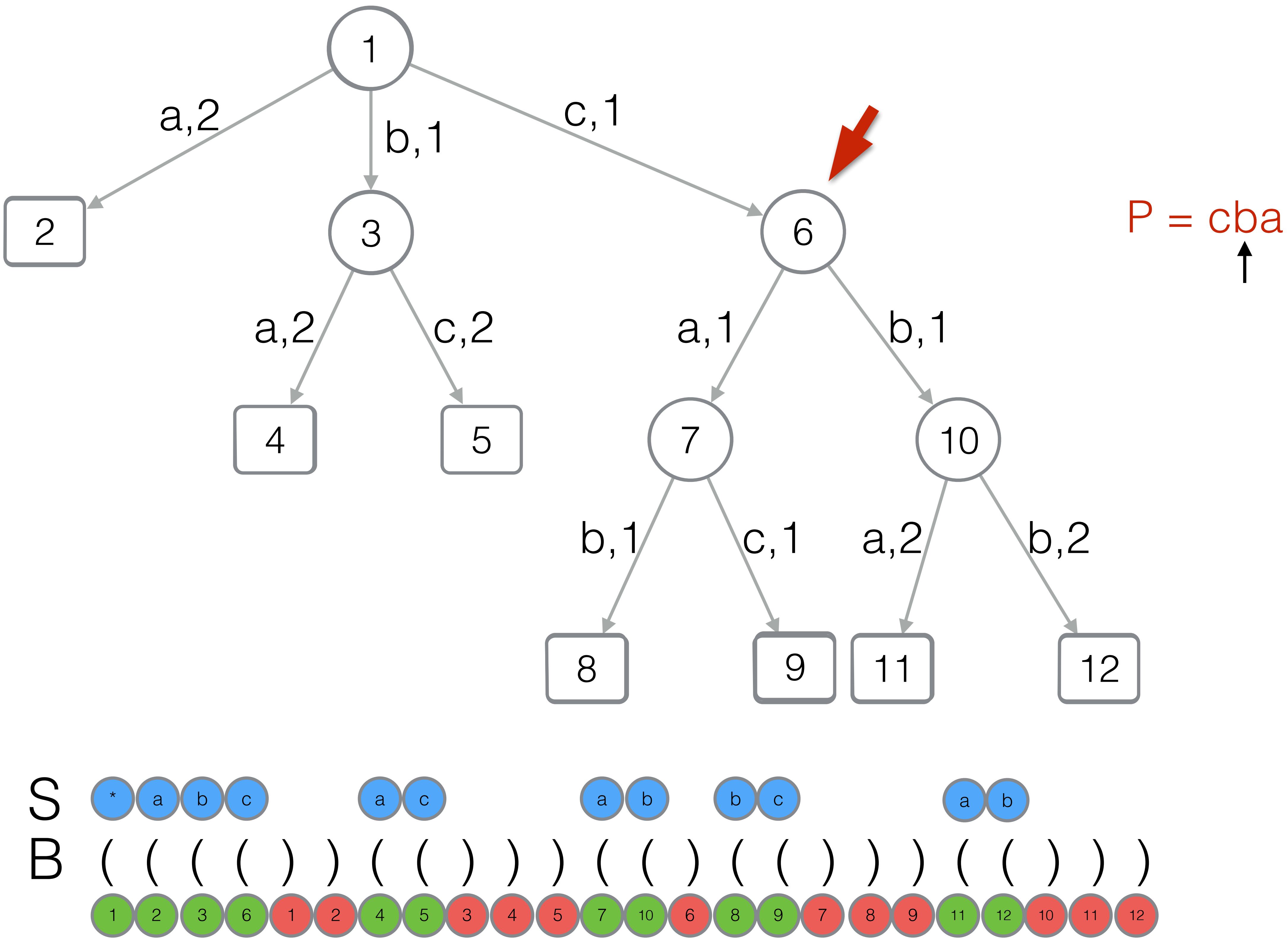
Patricia trie with DFUDS



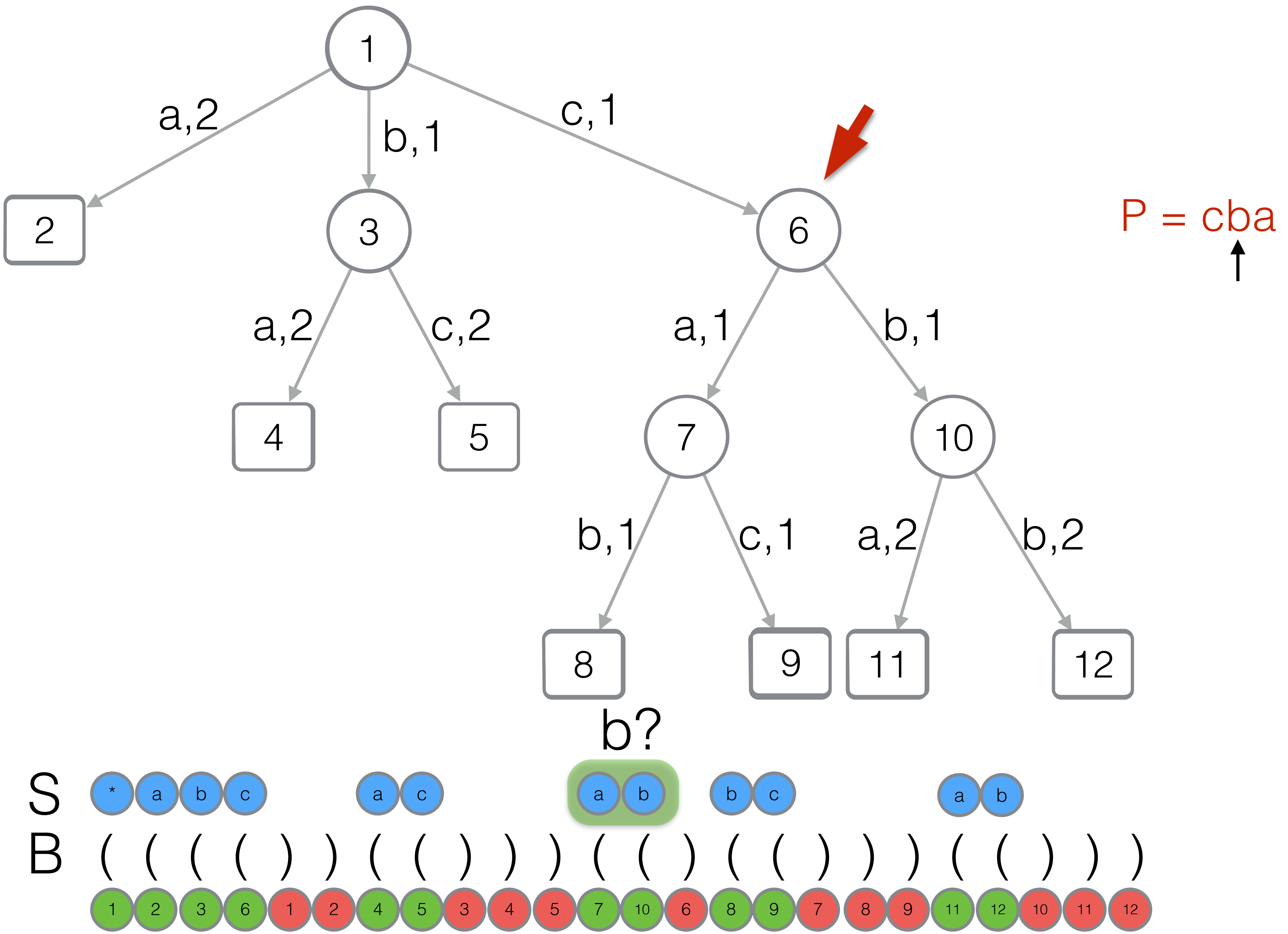
Patricia trie with DFUDS



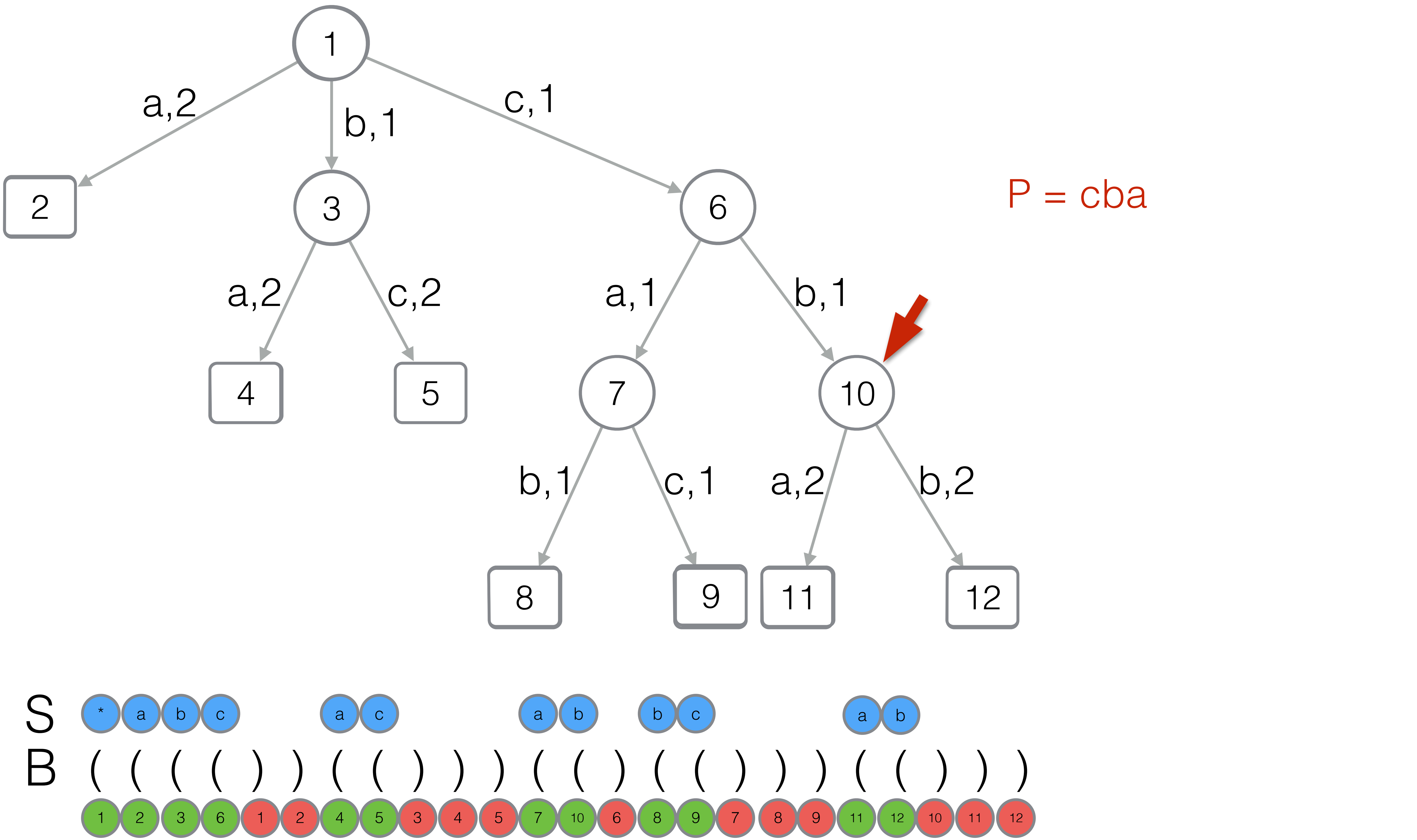
Patricia trie with DFUDS



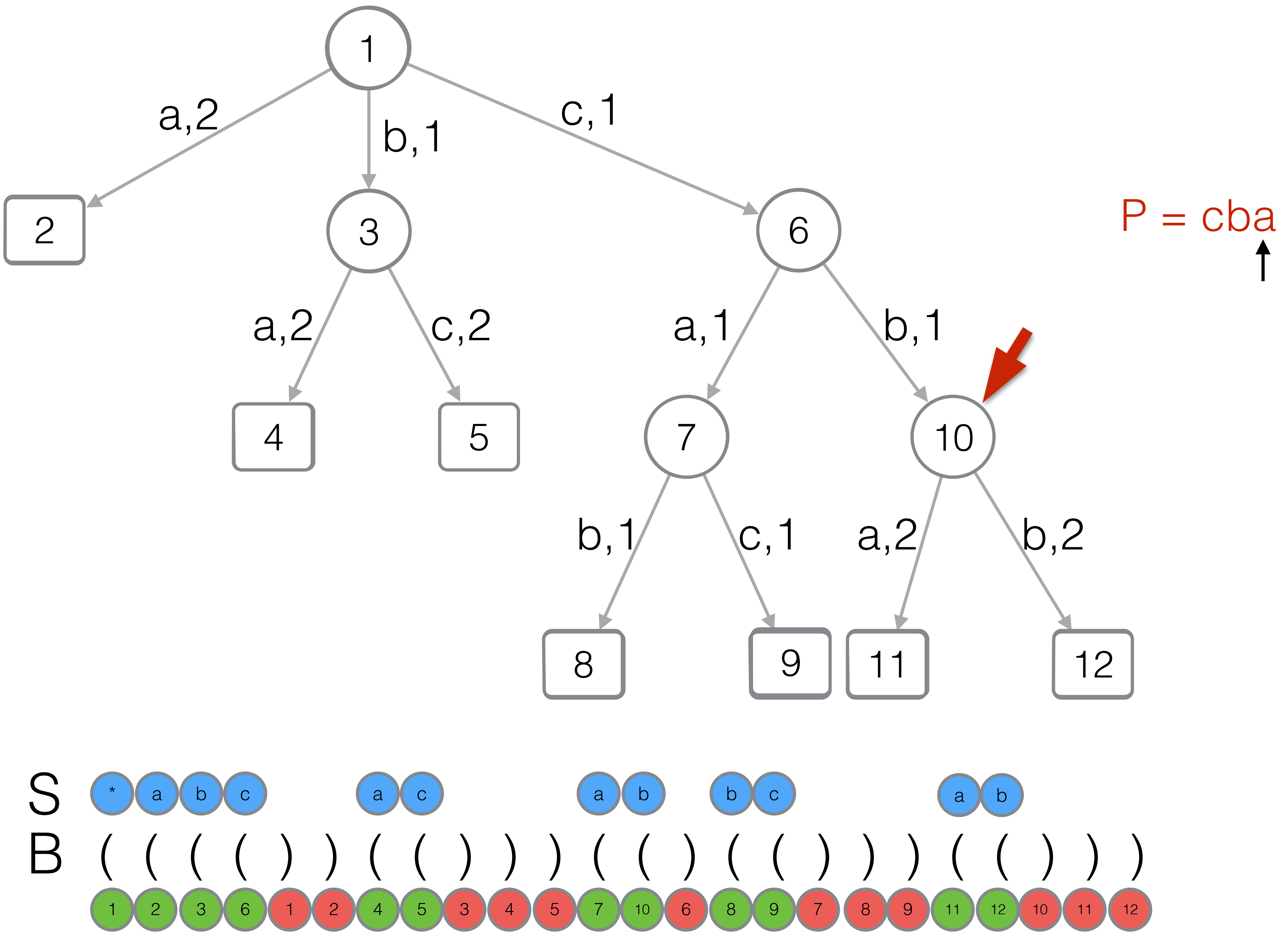
Patricia trie with DFUDS



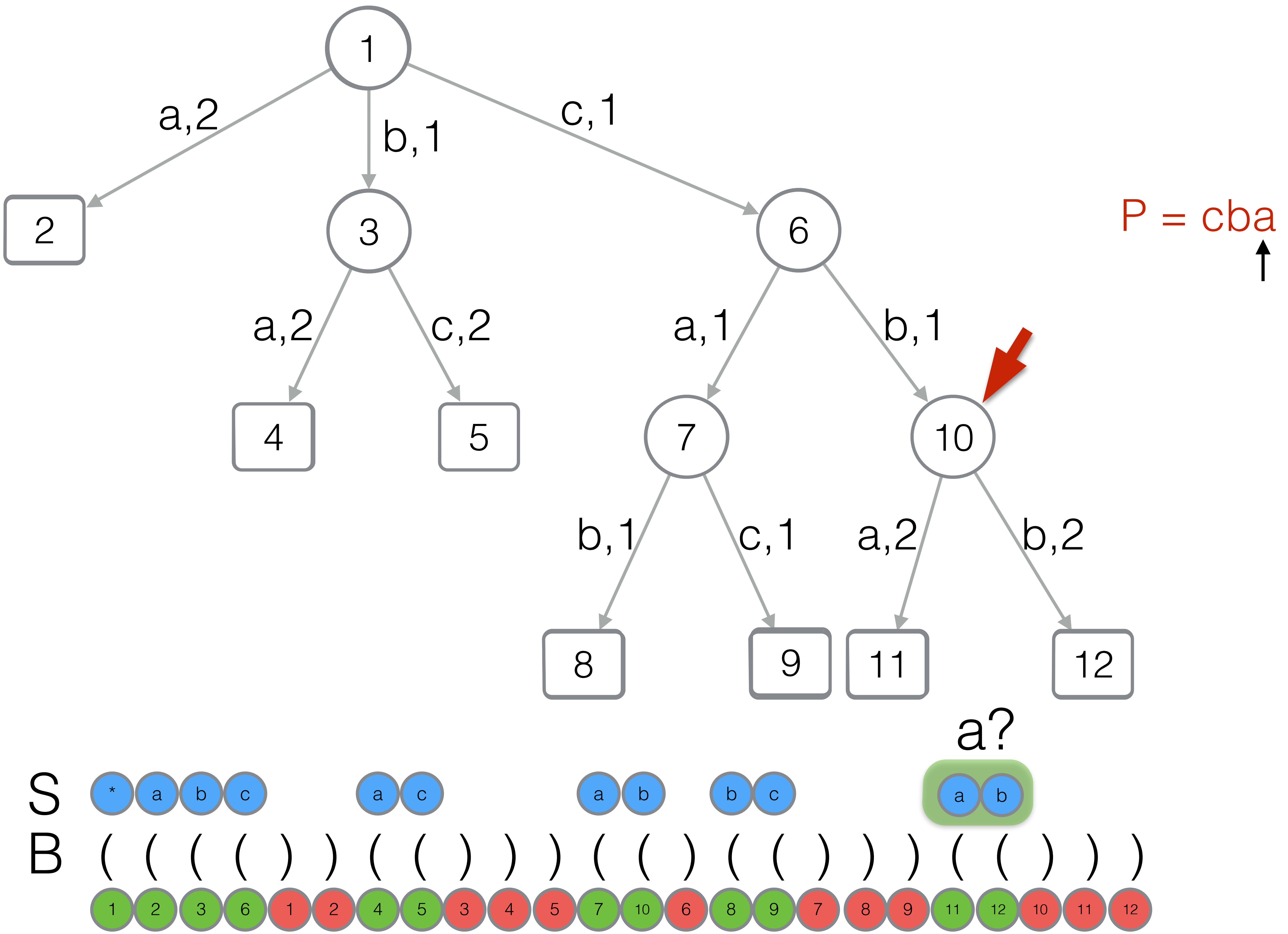
Patricia trie with DFUDS



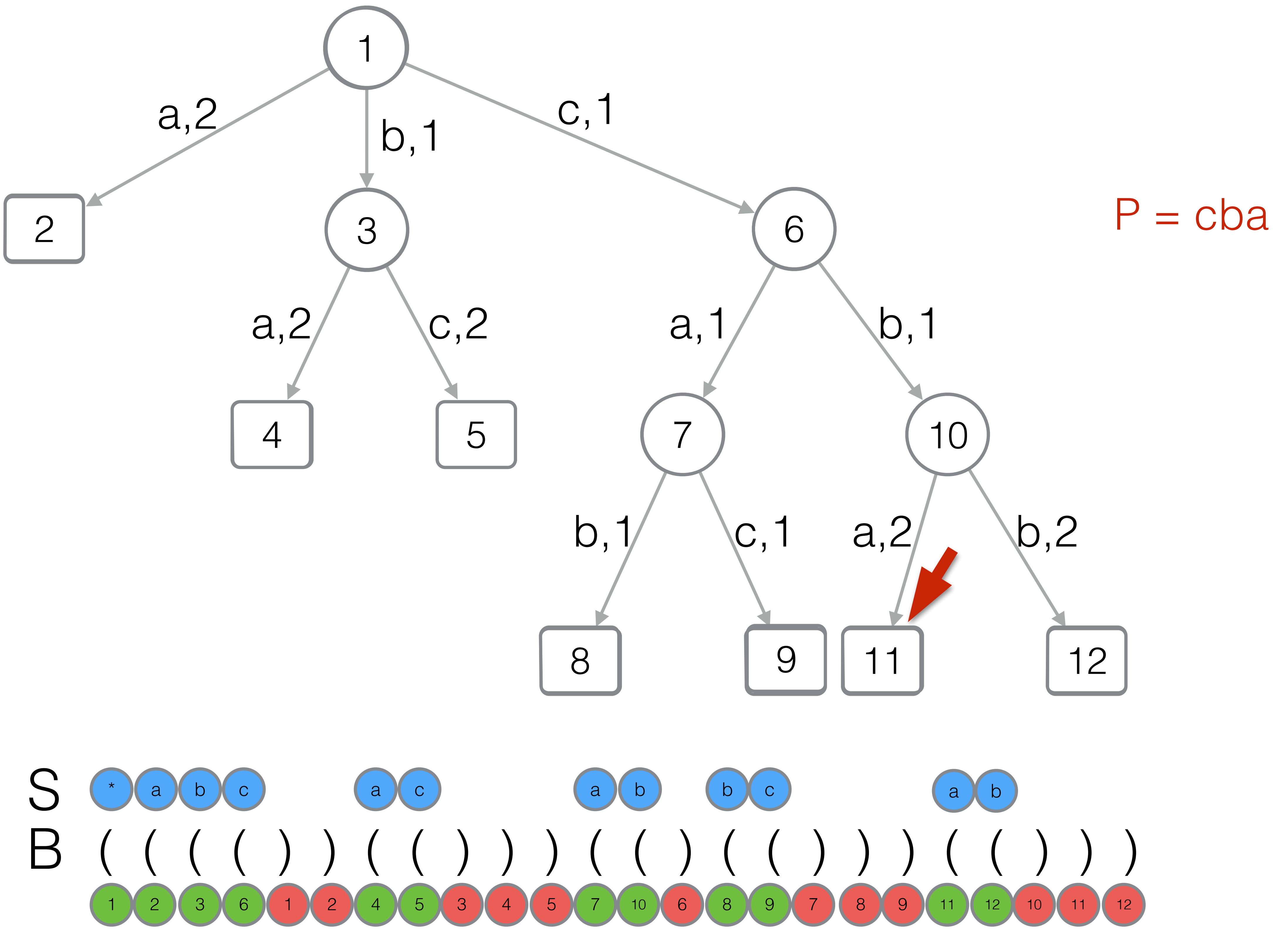
Patricia trie with DFUDS



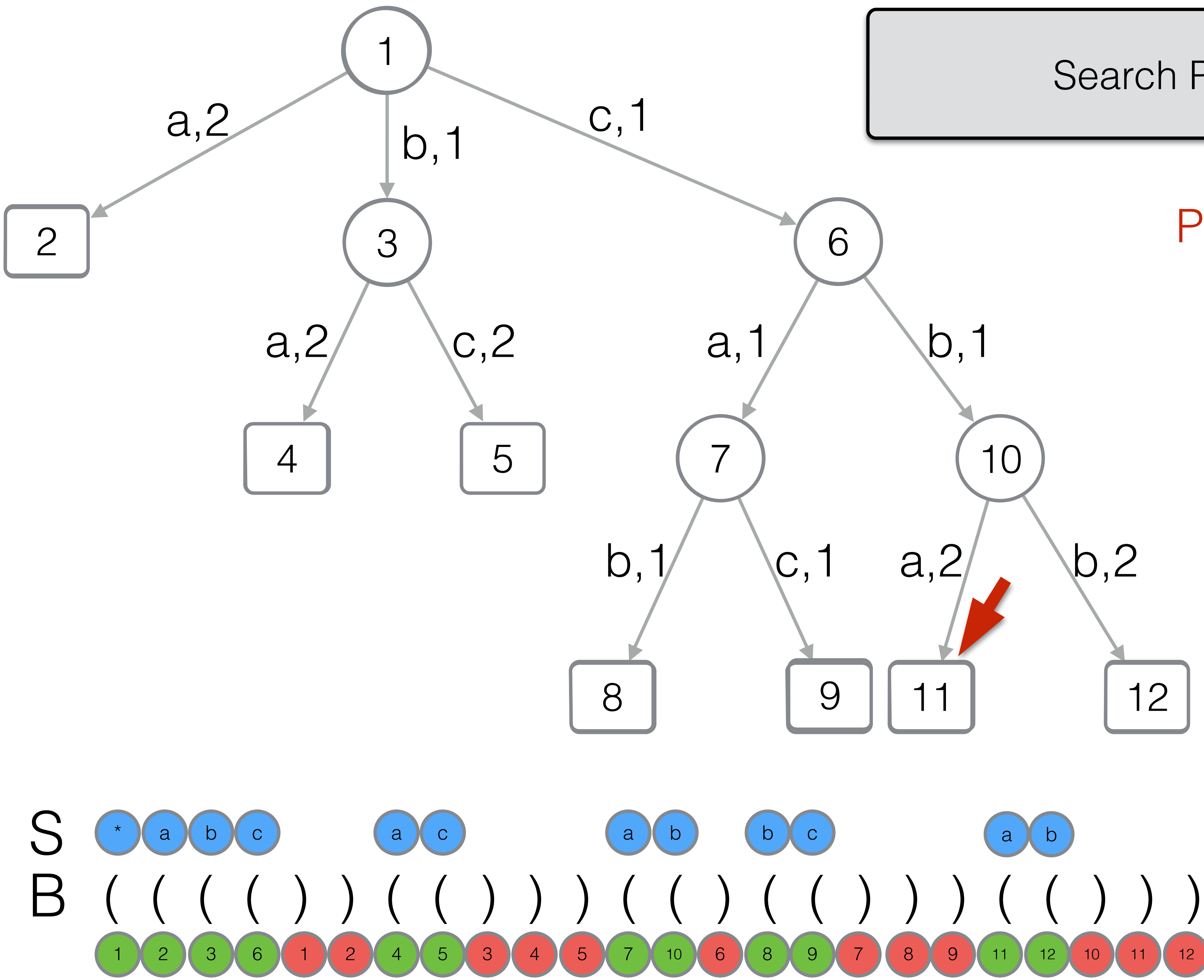
Patricia trie with DFUDS



Patricia trie with DFUDS



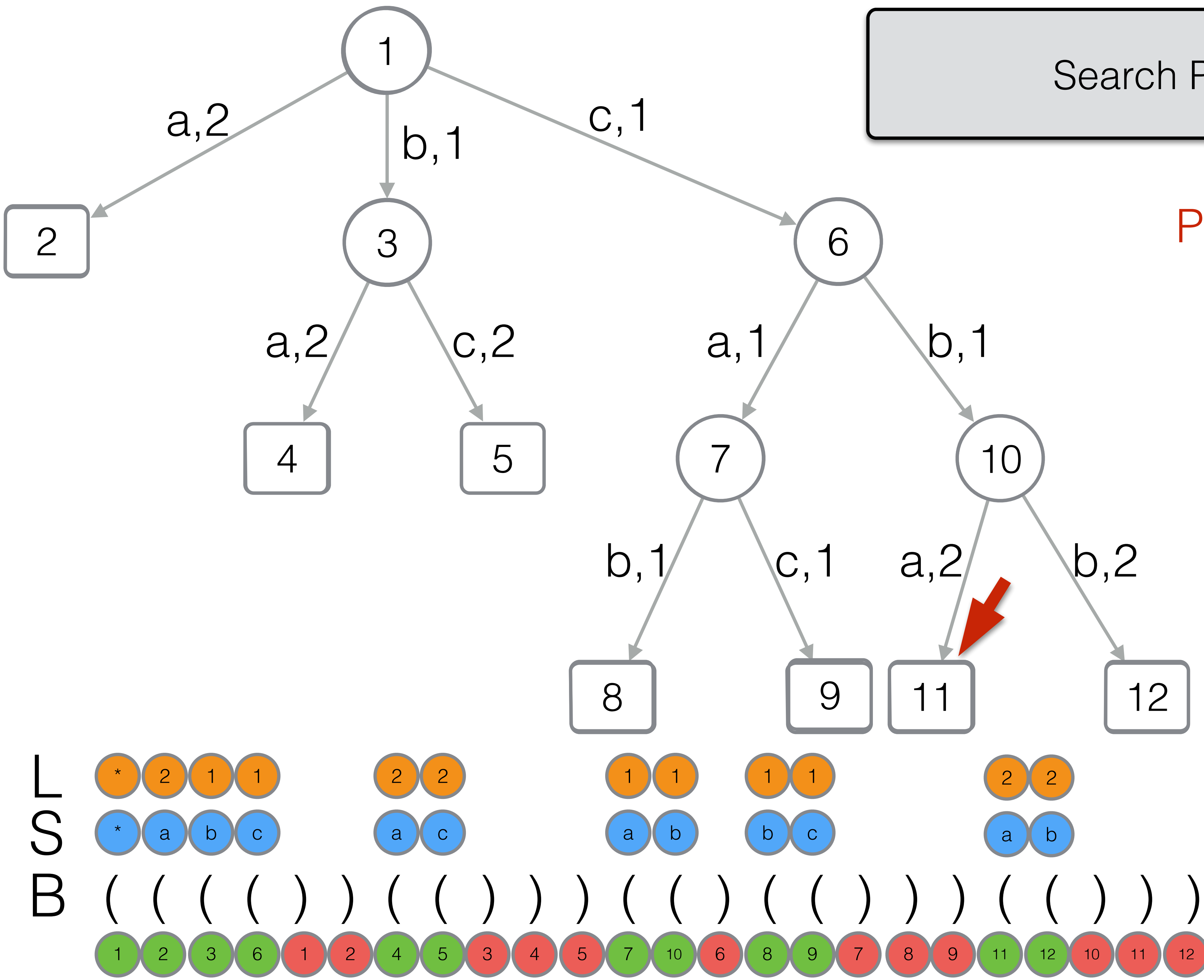
Patricia trie with DFUDS



Search P in O(|P|) time!

P = cba

Patricia trie with DFUDS



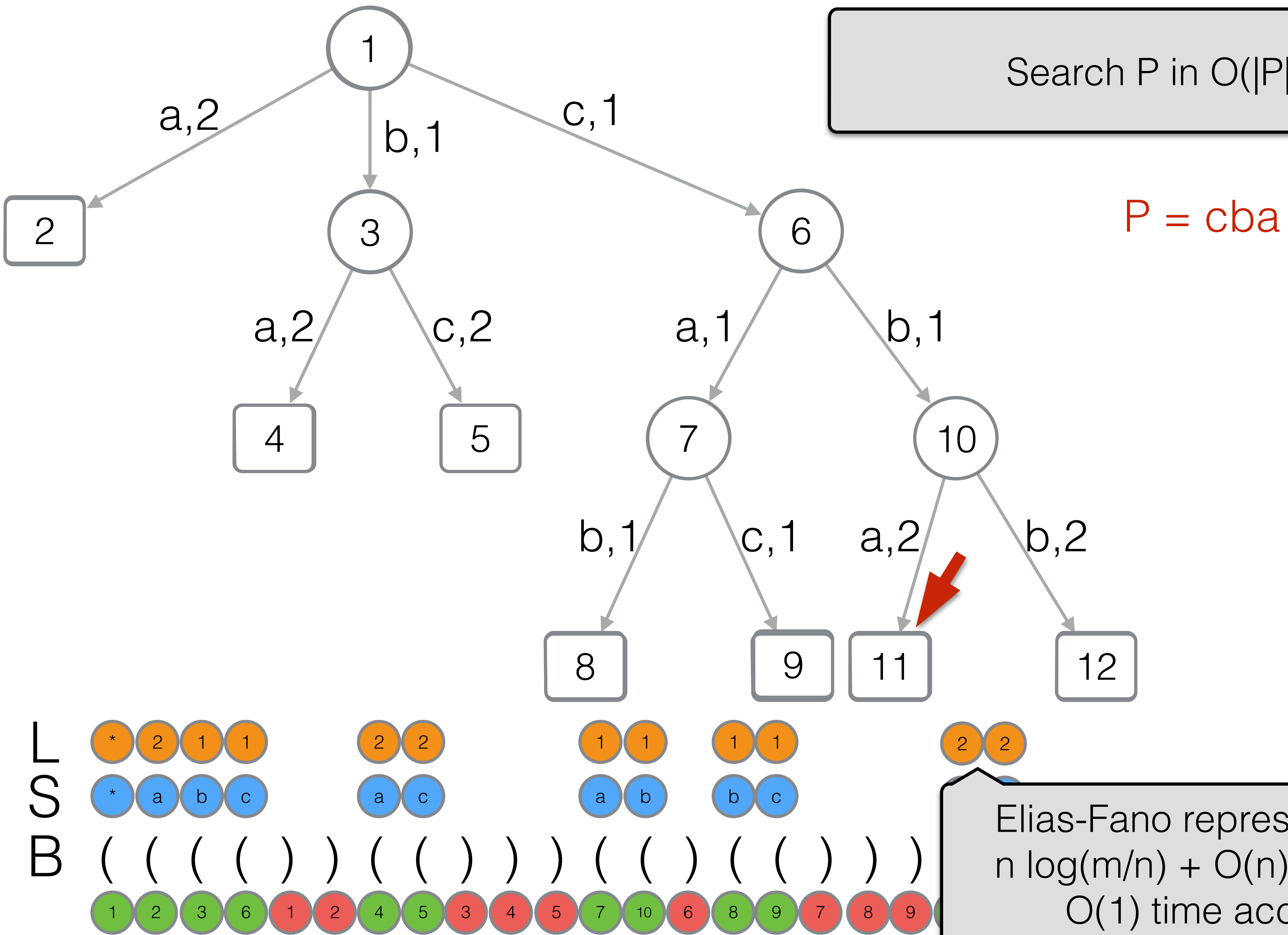
Search P in O(|P|) time!

P = cba

Patricia trie with DFUDS

Search P in O(|P|) time!

P = cba



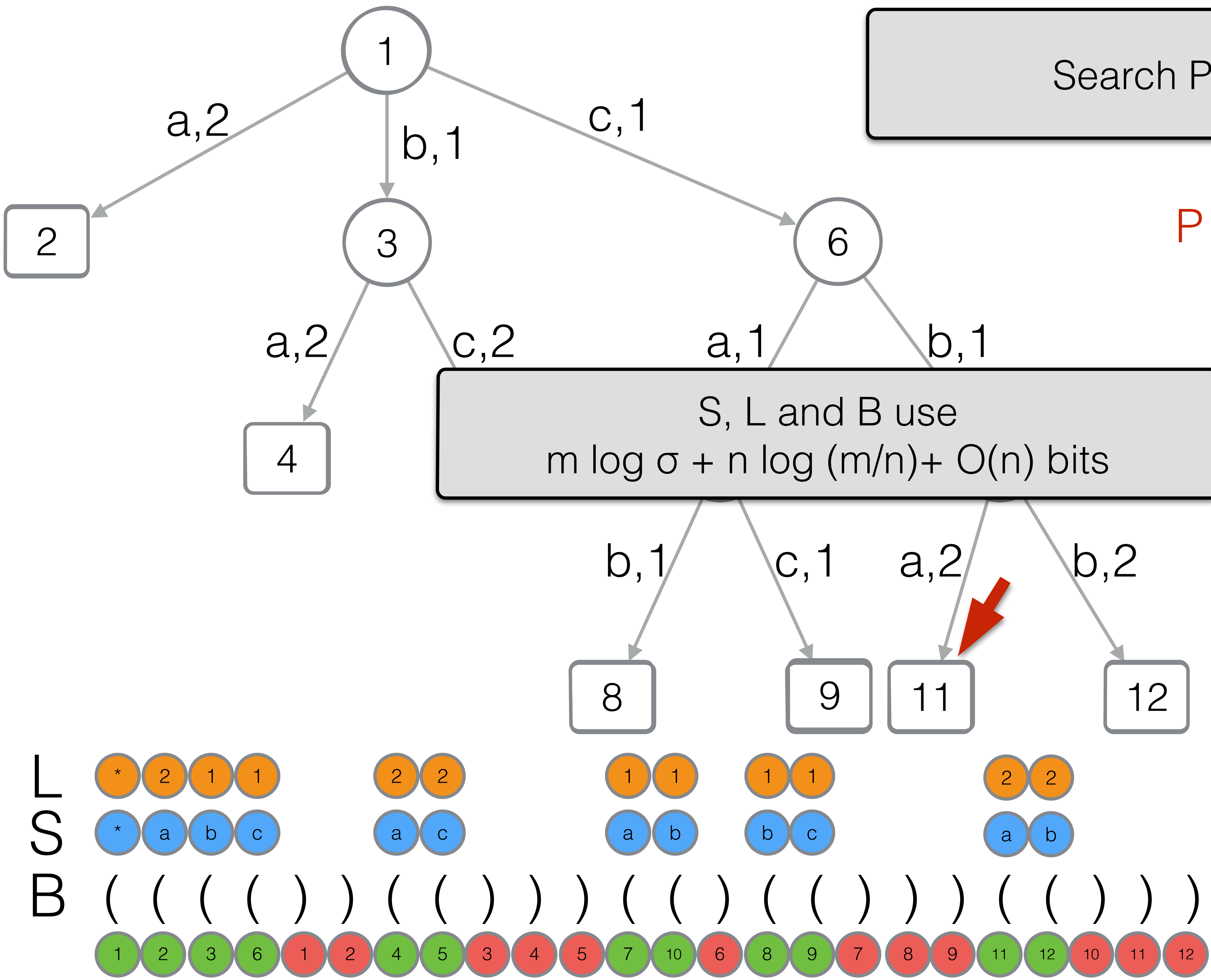
Elias-Fano representation: n log(m/n) + O(n) bits and O(1) time access.

Patricia trie with DFUDS

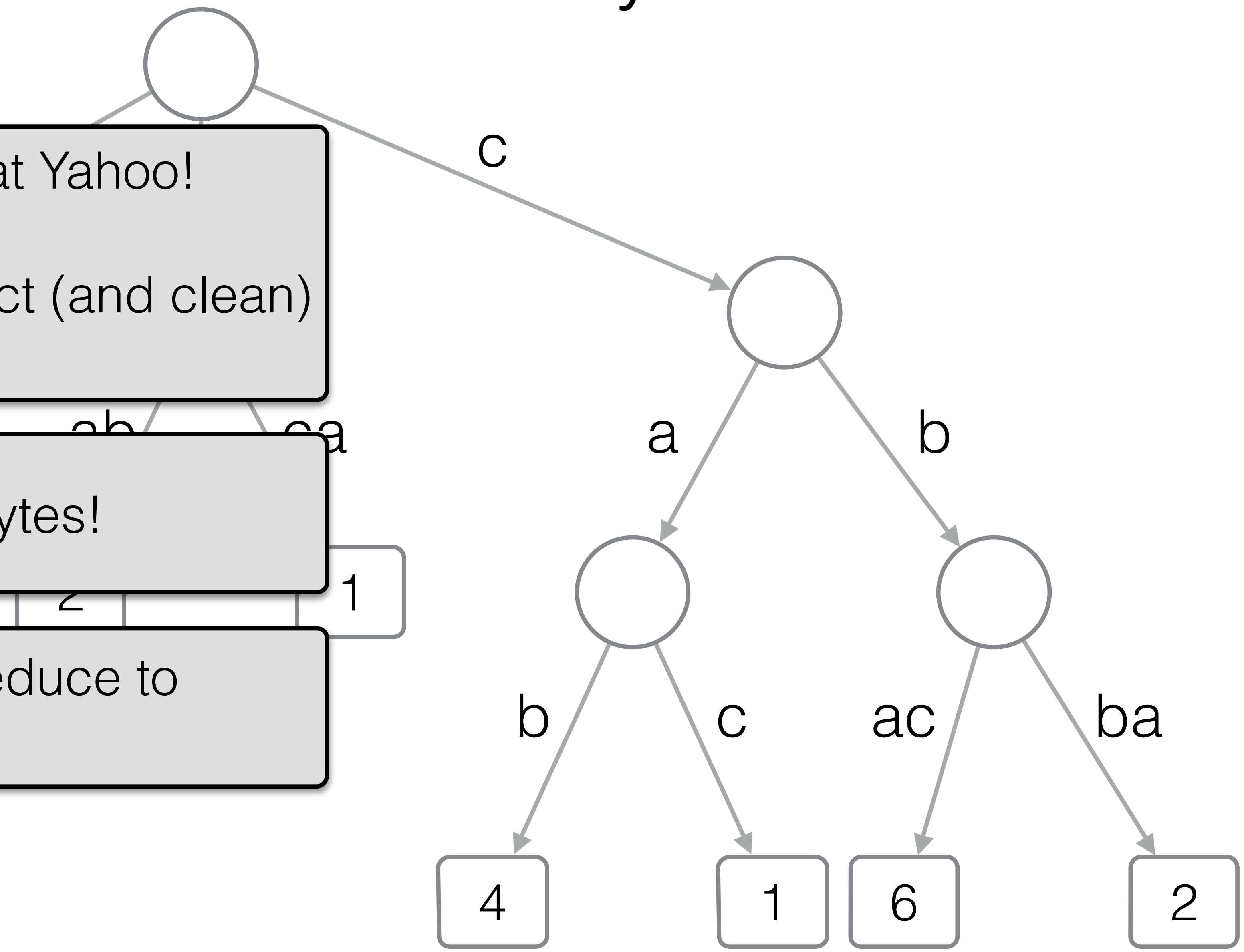
Search P in O(|P|) time!

P = cba

S, L and B use
$m \log \sigma + n \log (m/n) + O(n)$ bits



Summary



3 months query log at Yahoo!

≈600 million of distinct (and clean) queries

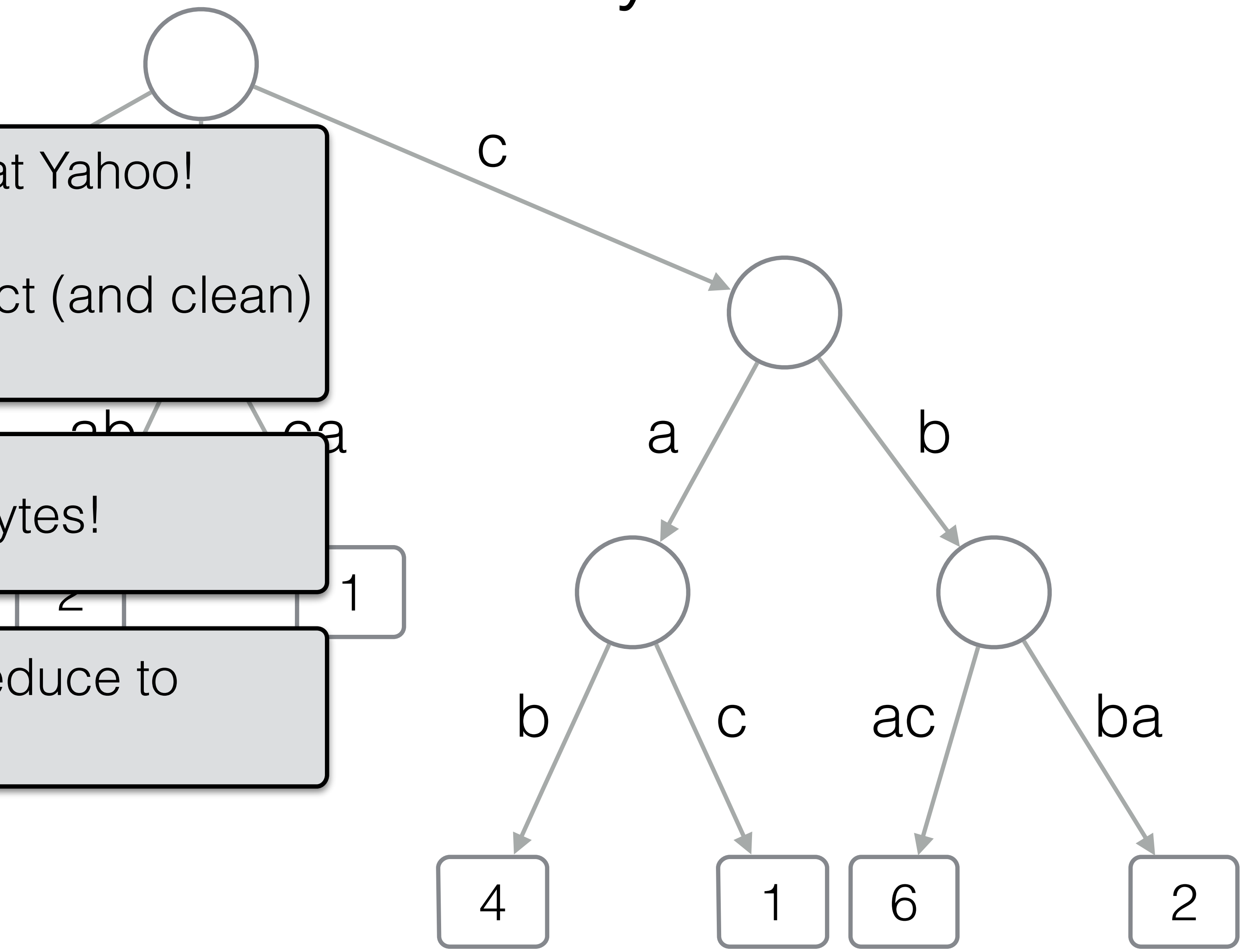
Trie requires ≈50 Gbytes!

We will see how to reduce to ≈5 Gbytes!

Find the node "prefixed" by P	O(|P|) time	
Compute the top-k strings	O(k log k) time	O(n) bits

n = |D|, m total length of strings in D

Summary



3 months query log at Yahoo!

≈600 million of distinct (and clean) queries

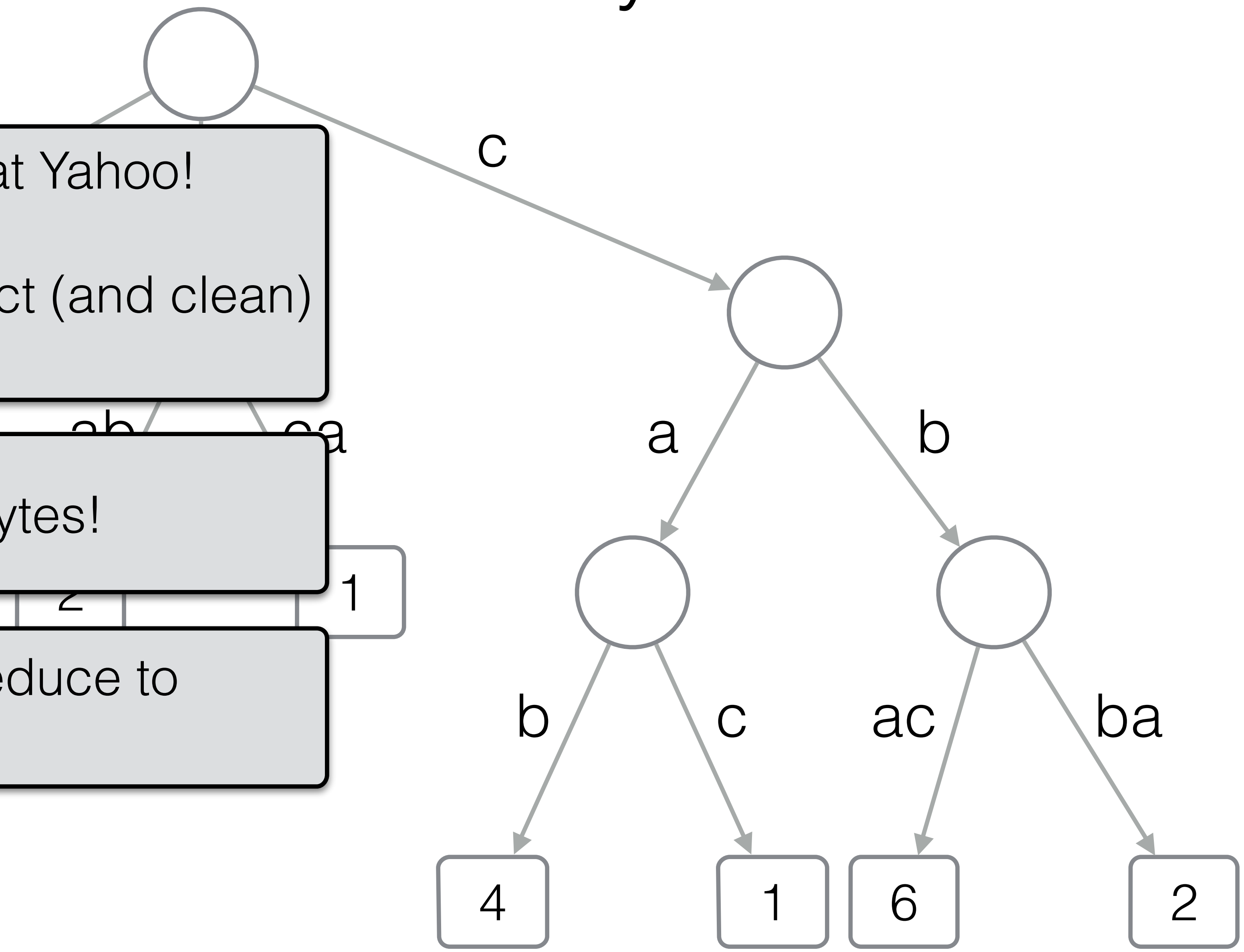
Trie requires ≈50 Gbytes!

We will see how to reduce to ≈5 Gbytes!

Find the node “prefixed” by P	O(|P|) time	m log σ + n log (m/n) + O(n) bits
Compute the top-k strings	O(k log k) time	O(n) bits

n = |D|, m total length of strings in D

Summary



3 months query log at Yahoo!

≈600 million of distinct (and clean) queries

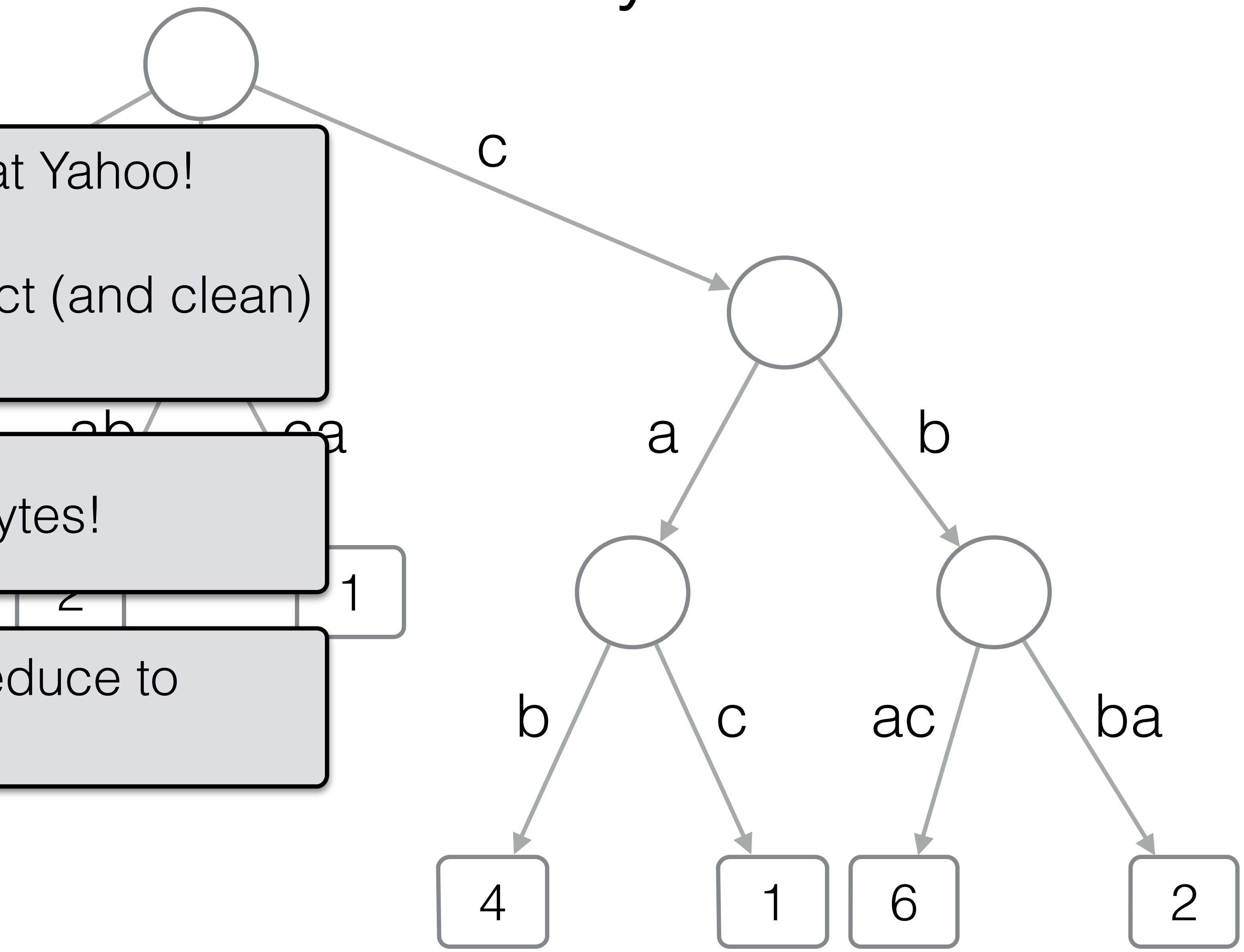
Trie requires ≈50 Gbytes!

We will see how to reduce to ≈5 Gbytes!

Find the node "prefixed" by P	O(|P|) time	$m \log \sigma + n \log (m/n) + O(n)$ bits
Compute the top-k strings	O(k log k) time	to be compared with $O(m \log \sigma + n \log m)$ bits

n = |D|, m total length of strings in D

Summary



3 months query log at Yahoo!

≈600 million of distinct (and clean) queries

Trie requires ≈50 Gbytes!

We will see how to reduce to ≈5 Gbytes!

Find the node “prefix

Compute the top-k st

(n=) 1 billion of strings of average length 64

$m=64*10^9$ symbols and $m/n = 64$

$m \log \sigma + n \log (m/n) + O(n)$ bits

to be compared with
$O(m \log \sigma + n \log m)$ bits

$n = |D|$, m total length of strings in D