Data Structure and Algorithms Design

Vincent Chau （周）

2021.11.24

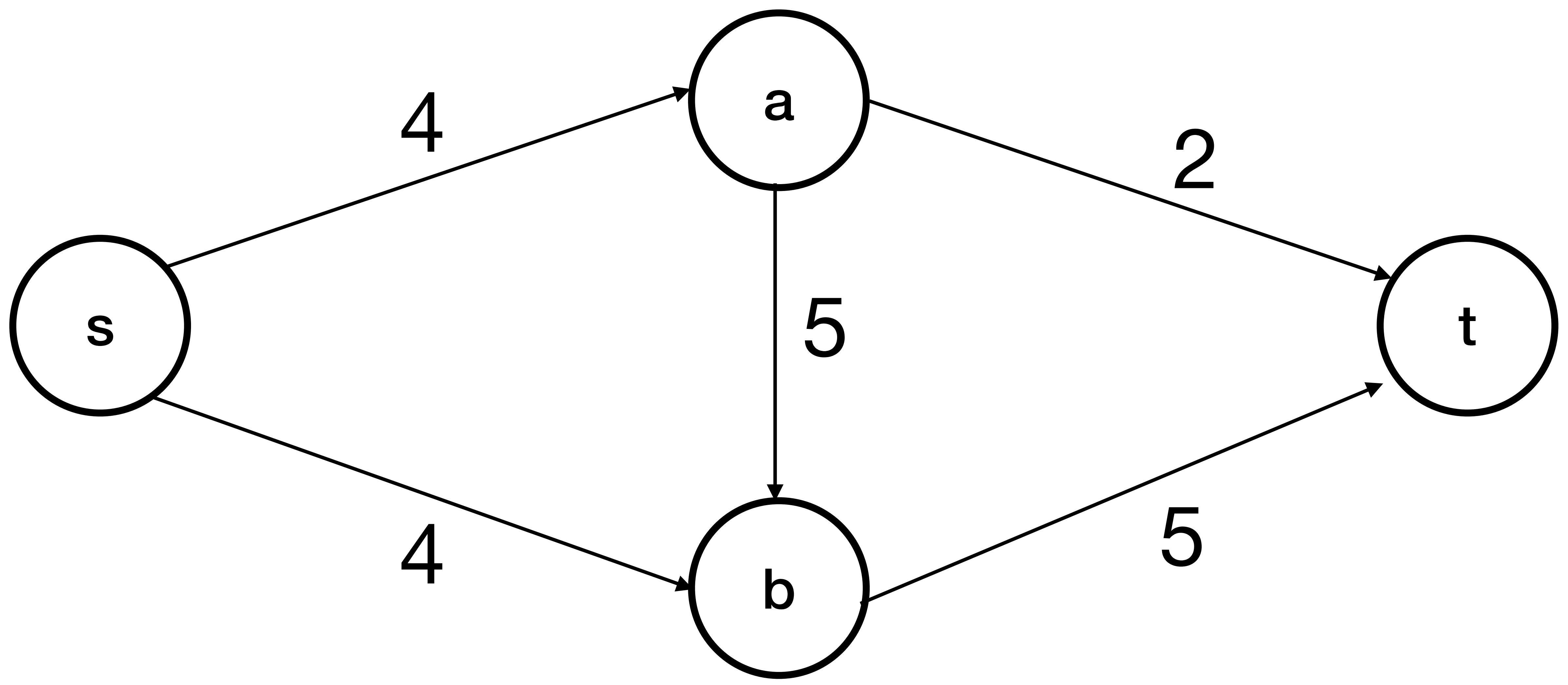
Summary

- Divide and Conquer √
- Dynamic programming √
- Greedy Algorithm √
- Binary Search Tree √
- Branch and Bound √
- Flow Networks
- Huffman Coding

Flow Network

Problem definition

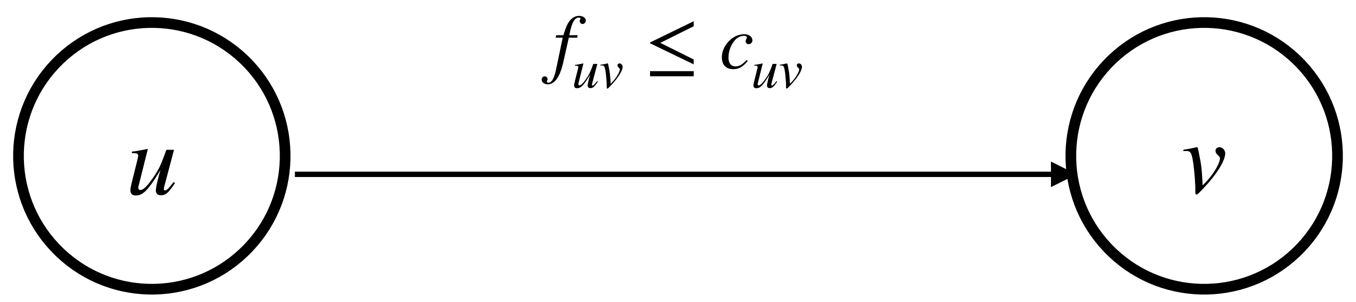
Given a direct graph G=(V,E), a source s and a sink t. Each edge in E has a capacity.



Goal: route as much flow as possible from the source s to the sink t

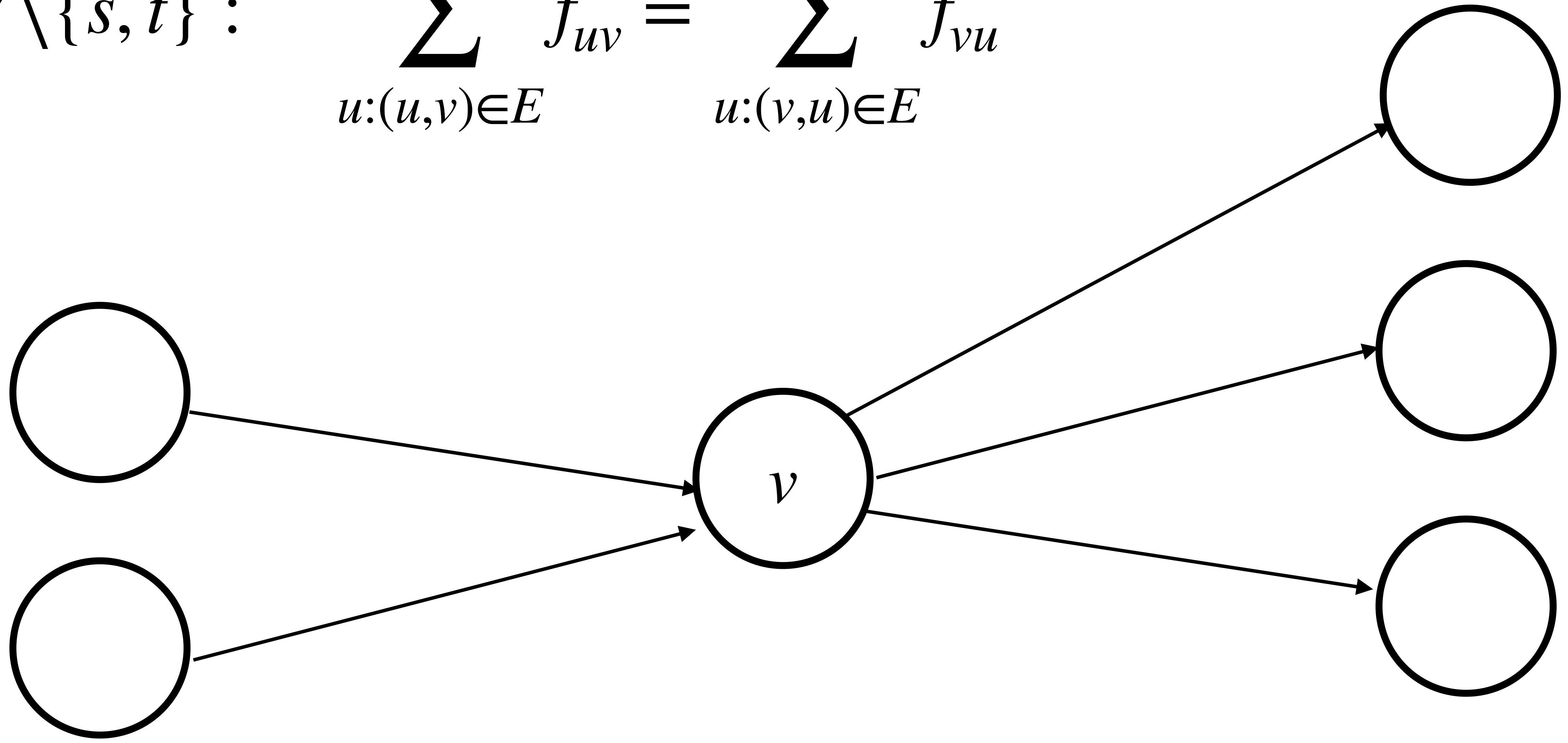
Feasible solution

- Capacity constraint: the flow of an edge cannot exceed its capacity $f_{uv} \leq c_{uv}\ \forall (u,v) \in E$

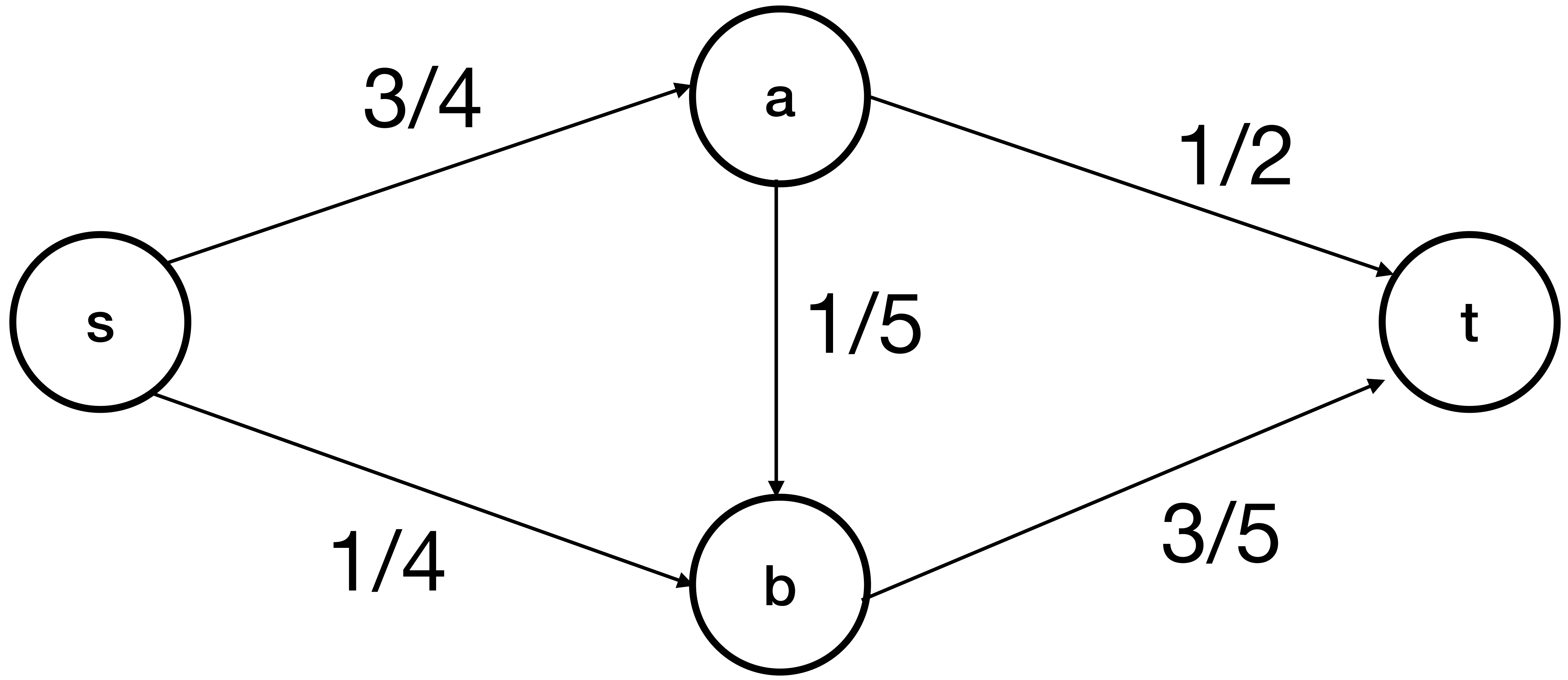


- Conservation of flows: the sum of the flows entering a node must equal the sum of the flows exiting that node, except for the source and the sink

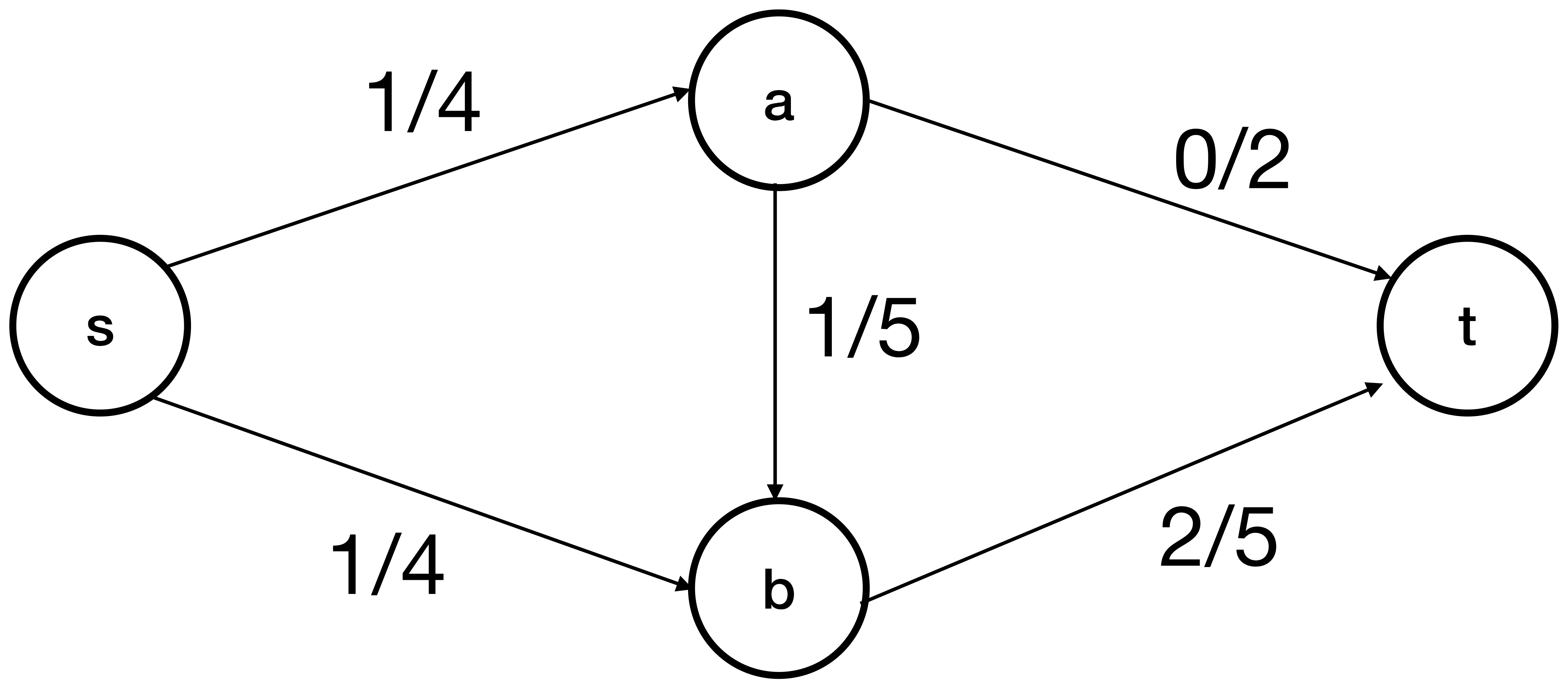
$$\forall v \in V \backslash \{s,t\} : \sum_{u:(u,v)\in E} f_{uv} = \sum_{u:(v,u)\in E} f_{vu}$$



Feasible solution?

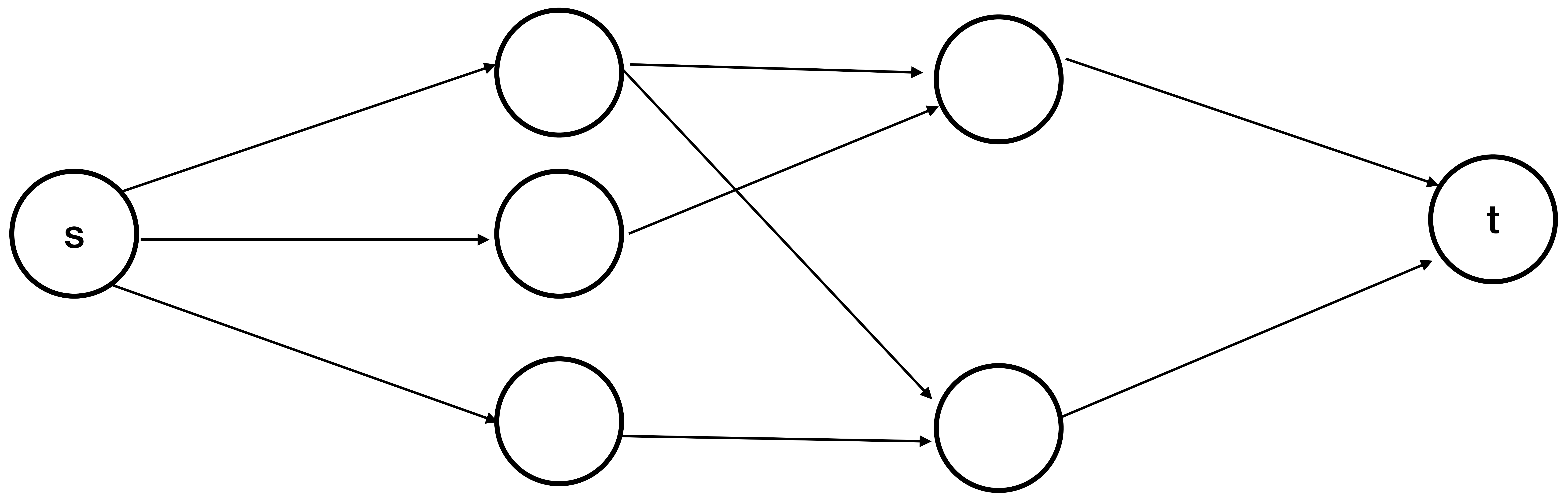


Feasible solution?



Application

- Transportation problems
- Maximum matching on bipartite graph



Taught algorithms

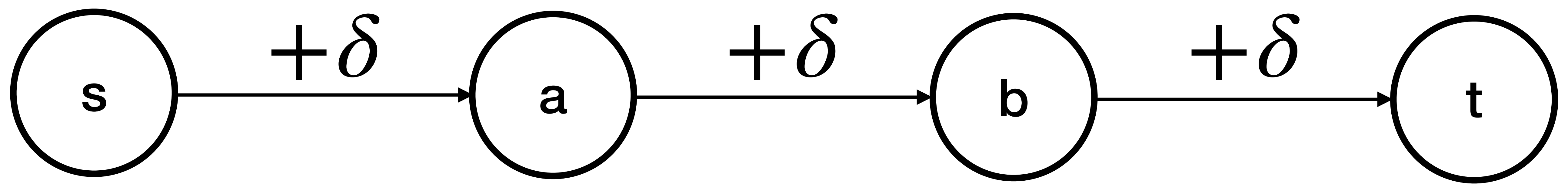
- Ford-Fulkerson
- Edmonds Karp

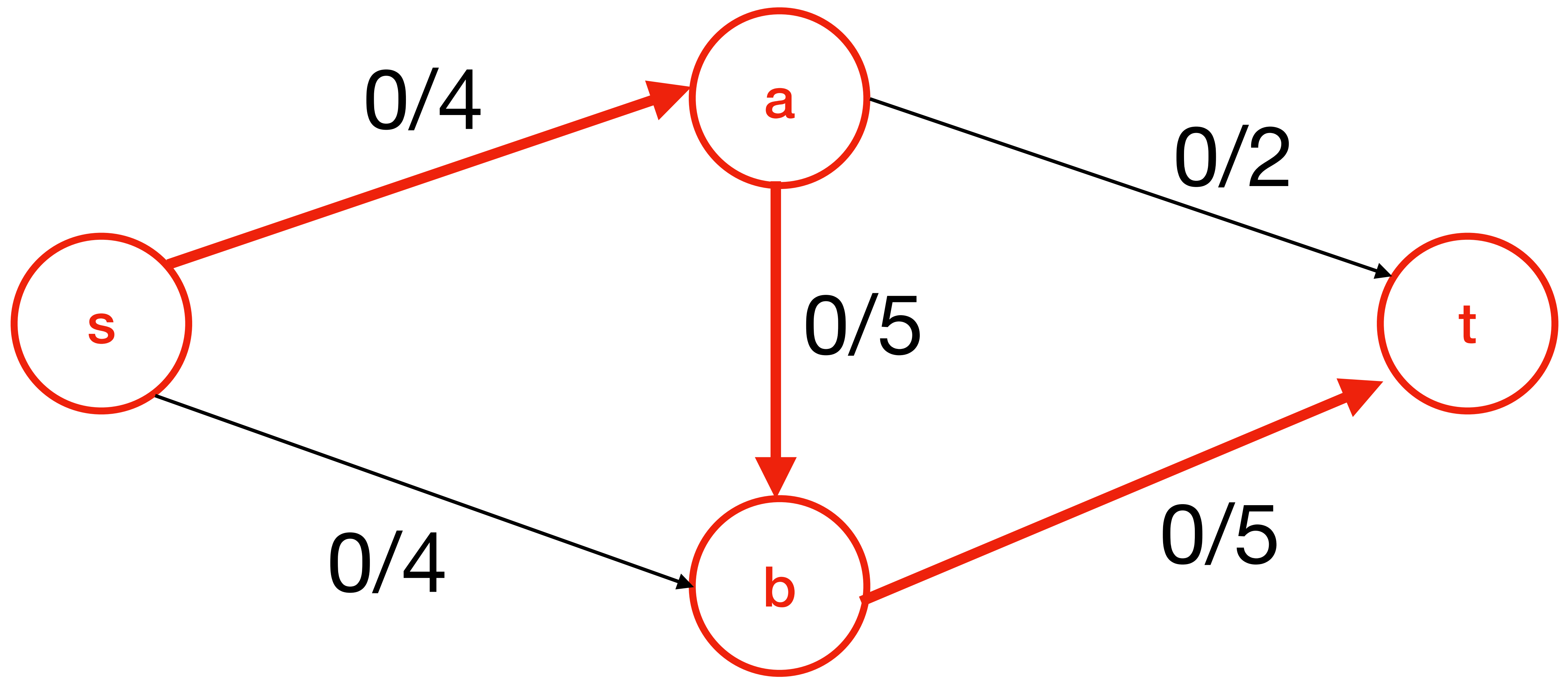
Ford-Fulkerson

- Set all flows to 0
- Find an augmenting path from s to t
- Add as much flow as possible flow
- Repeat the above whenever there is an augmenting path

Augmenting Path

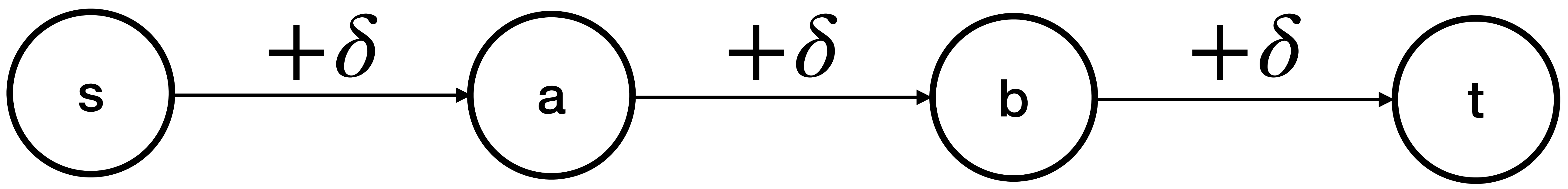
- A path from s to t such that the flow can be increased

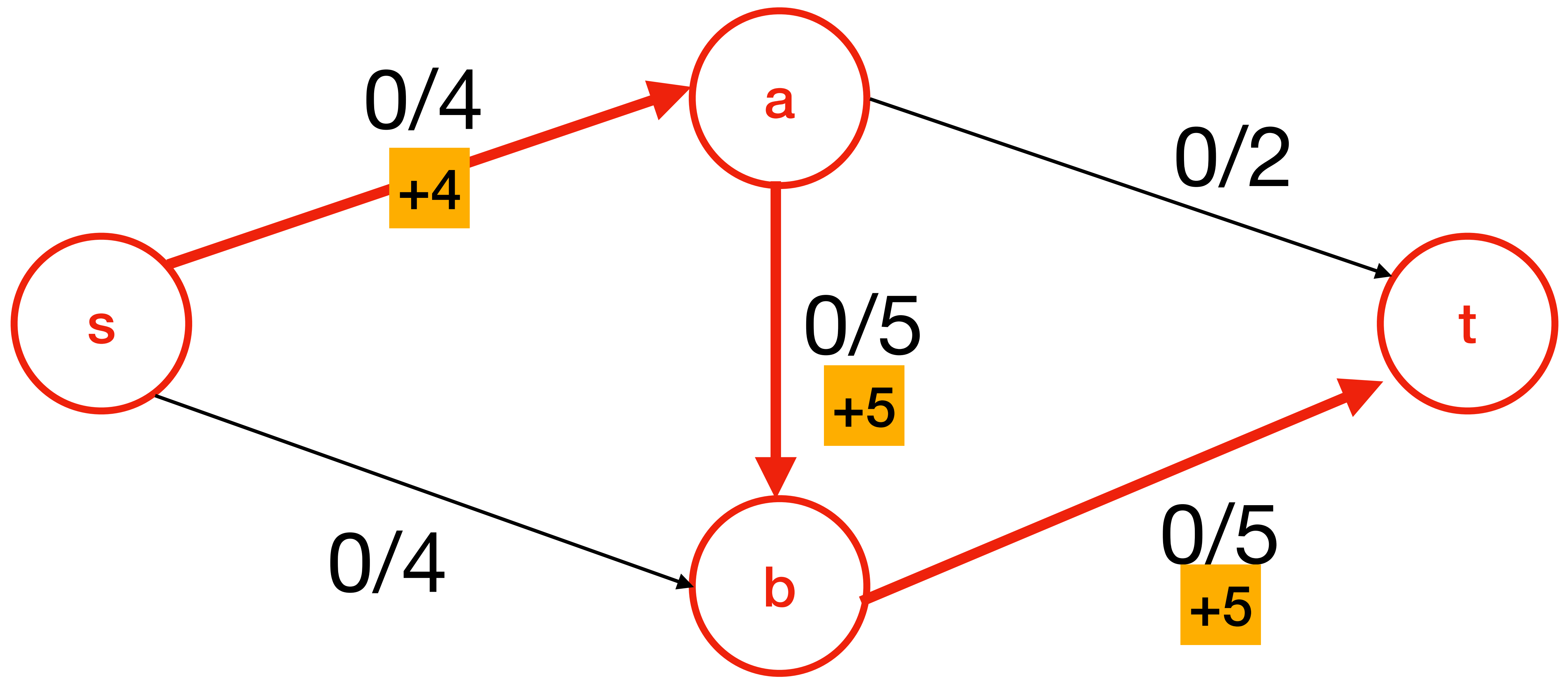




Augmenting Path

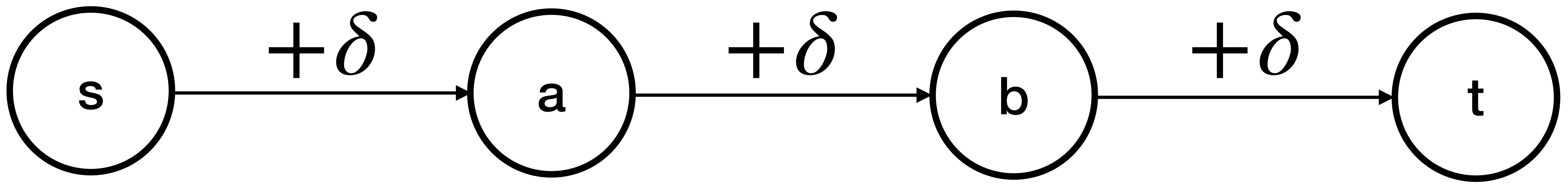
- A path from s to t such that the flow can be increased

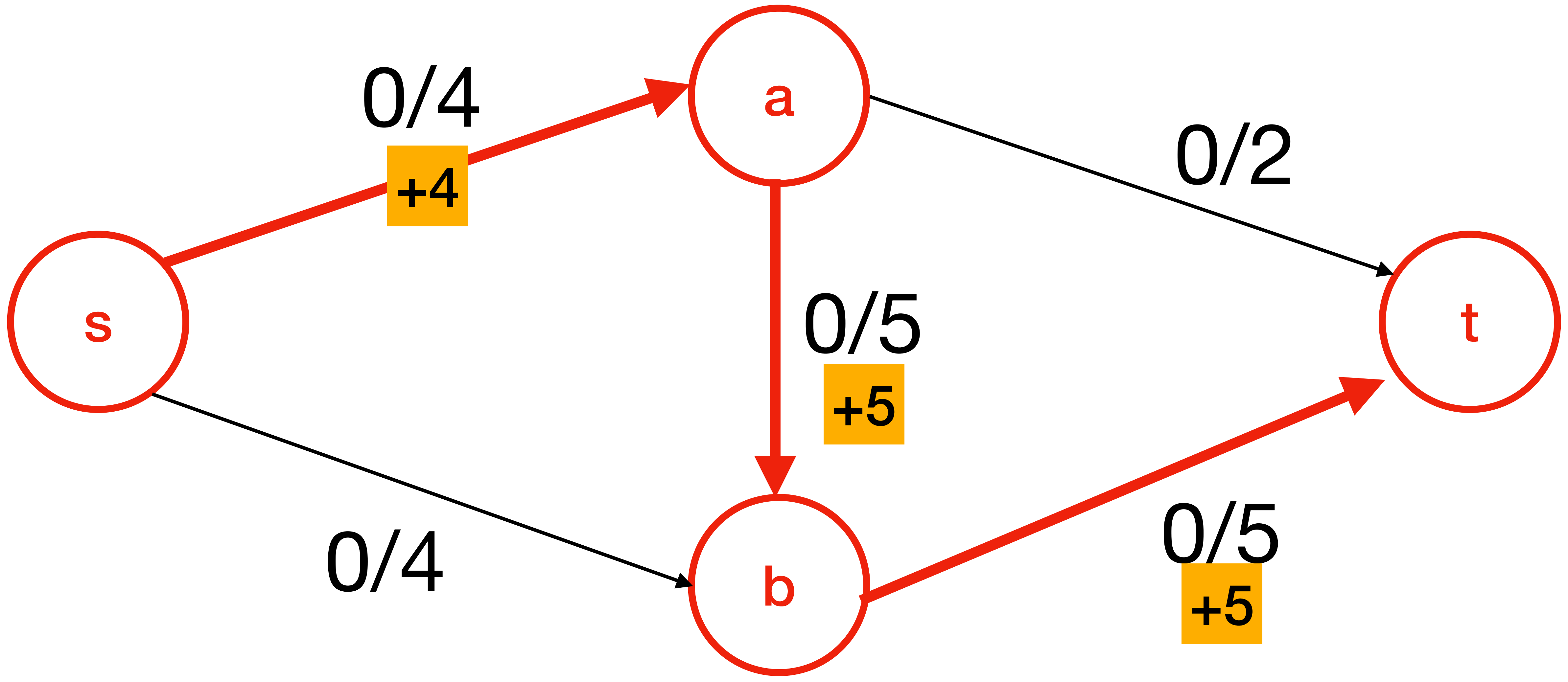




Augmenting Path

- A path from s to t such that the flow can be increased

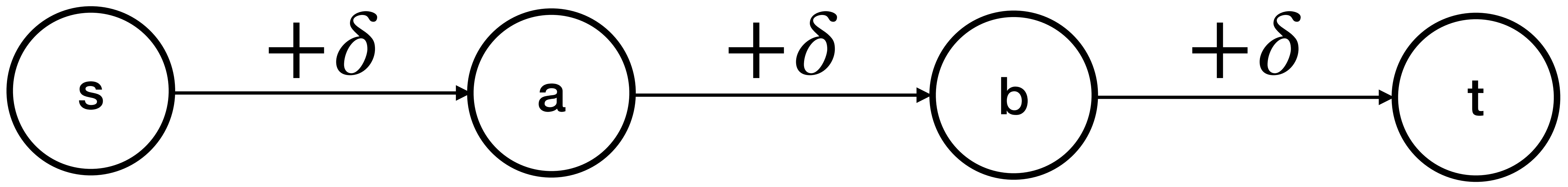


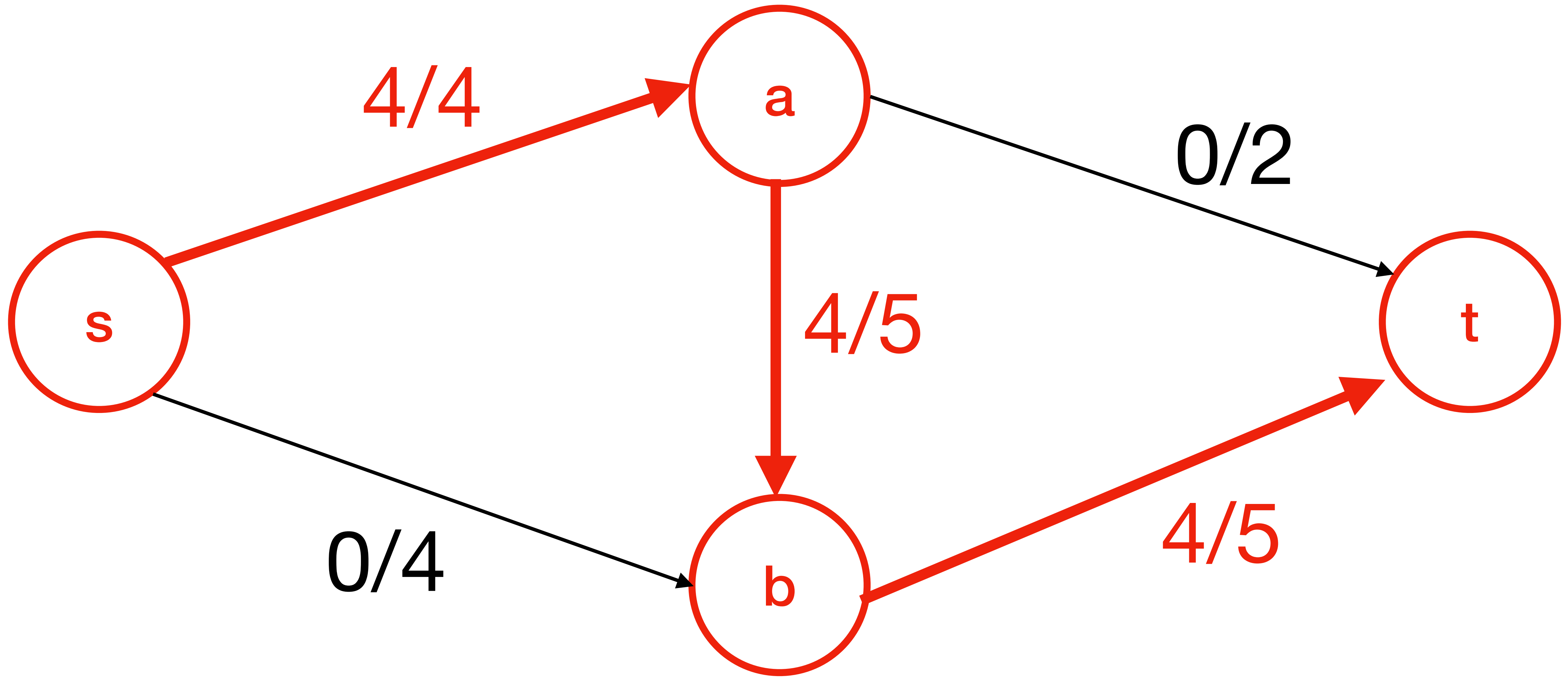


$$\delta = \min\{4,\ 5,\ 5\}$$

Augmenting Path

- A path from s to t such that the flow can be increased

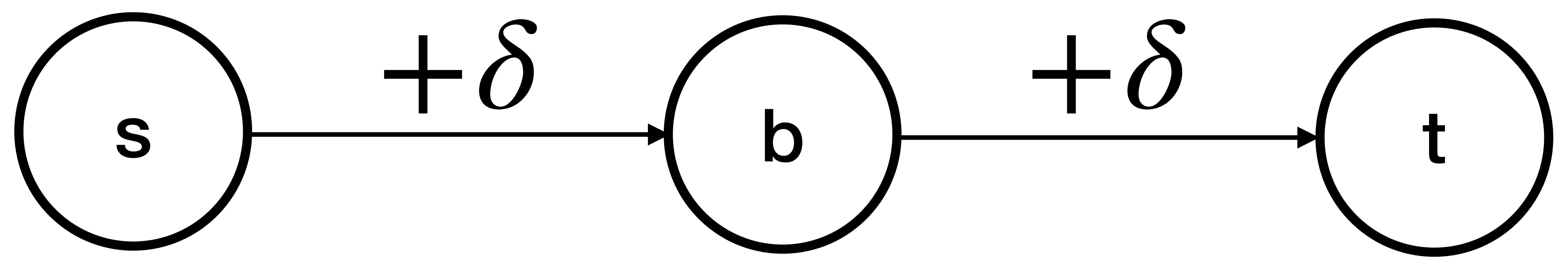


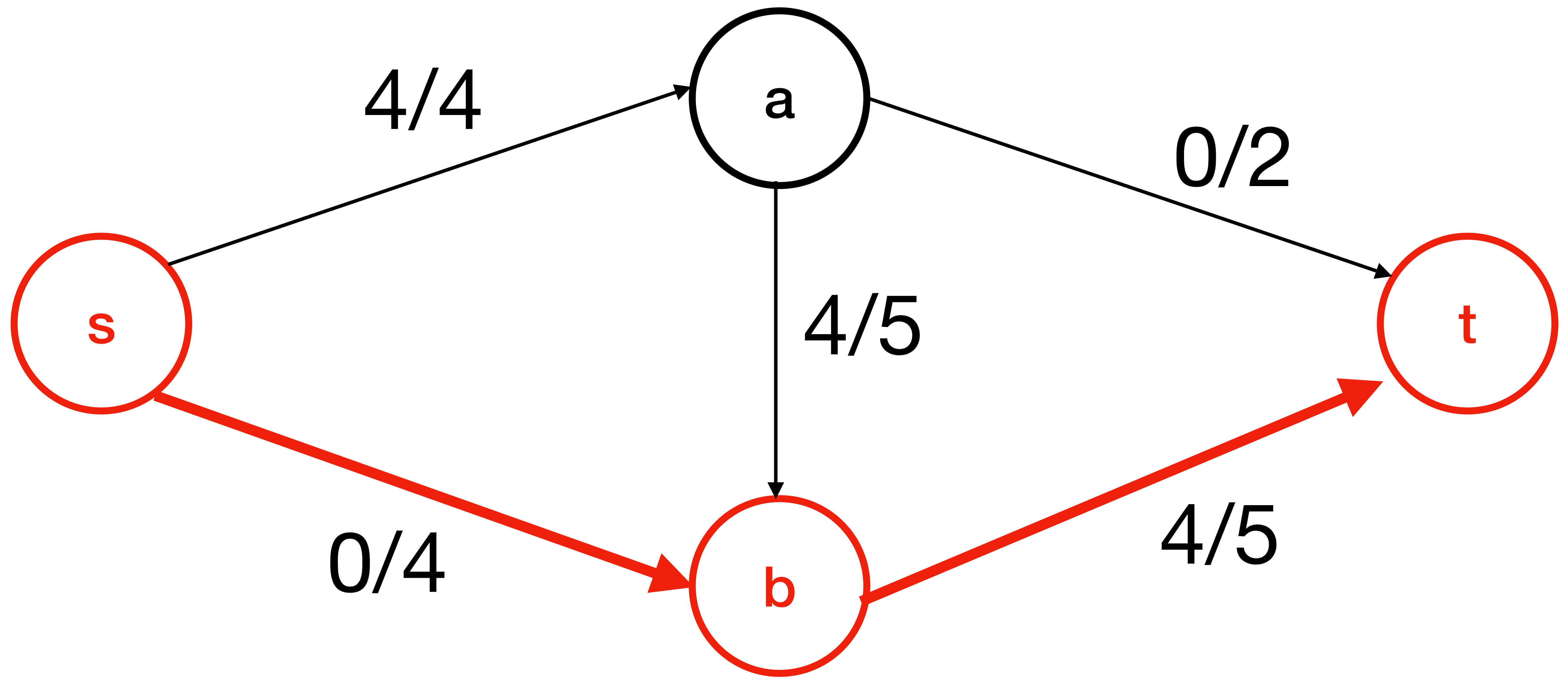


$$\delta = \min\{4, 5, 5\}$$

Augmenting Path

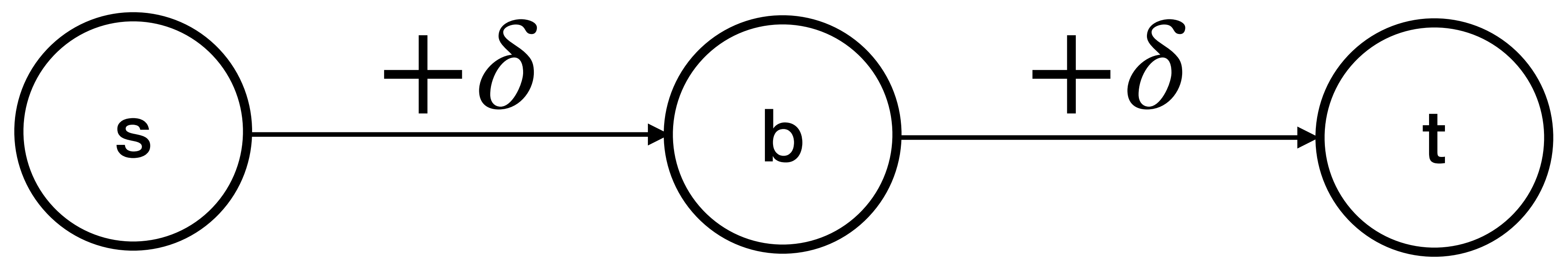
- A path from s to t such that the flow can be increased

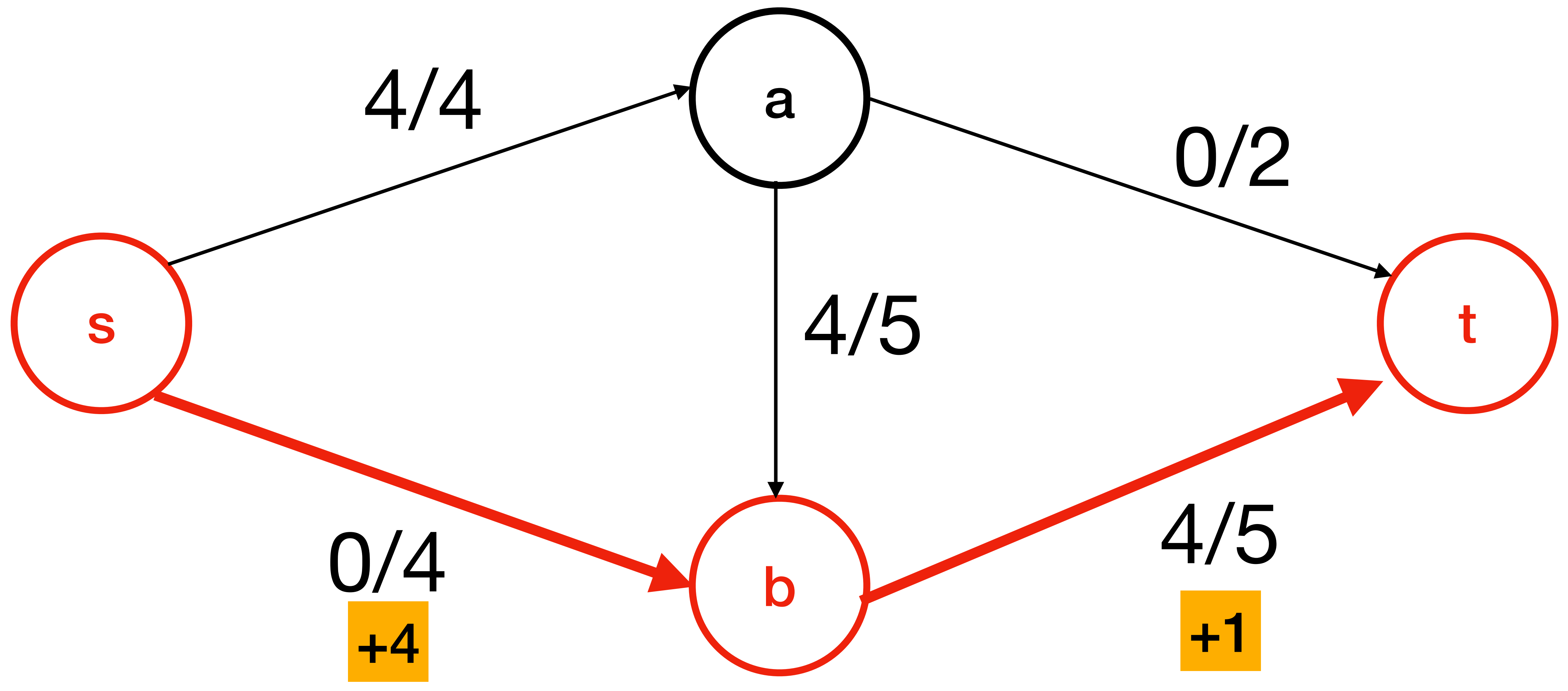




Augmenting Path

- A path from s to t such that the flow can be increased

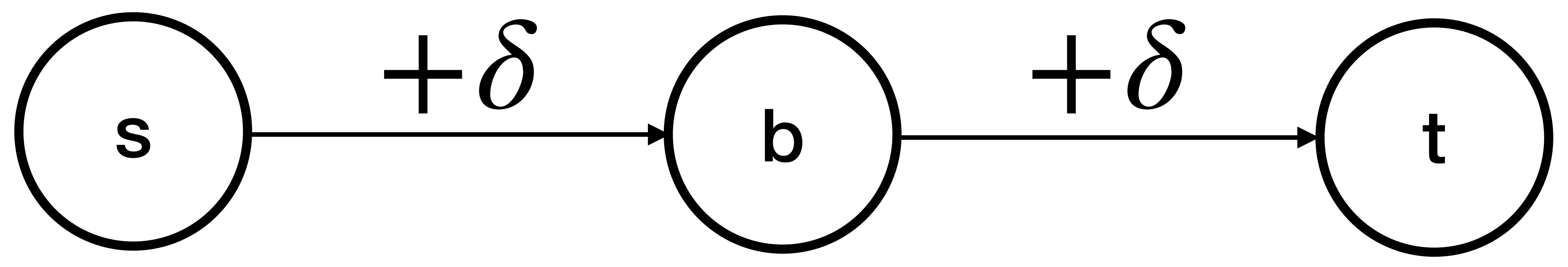


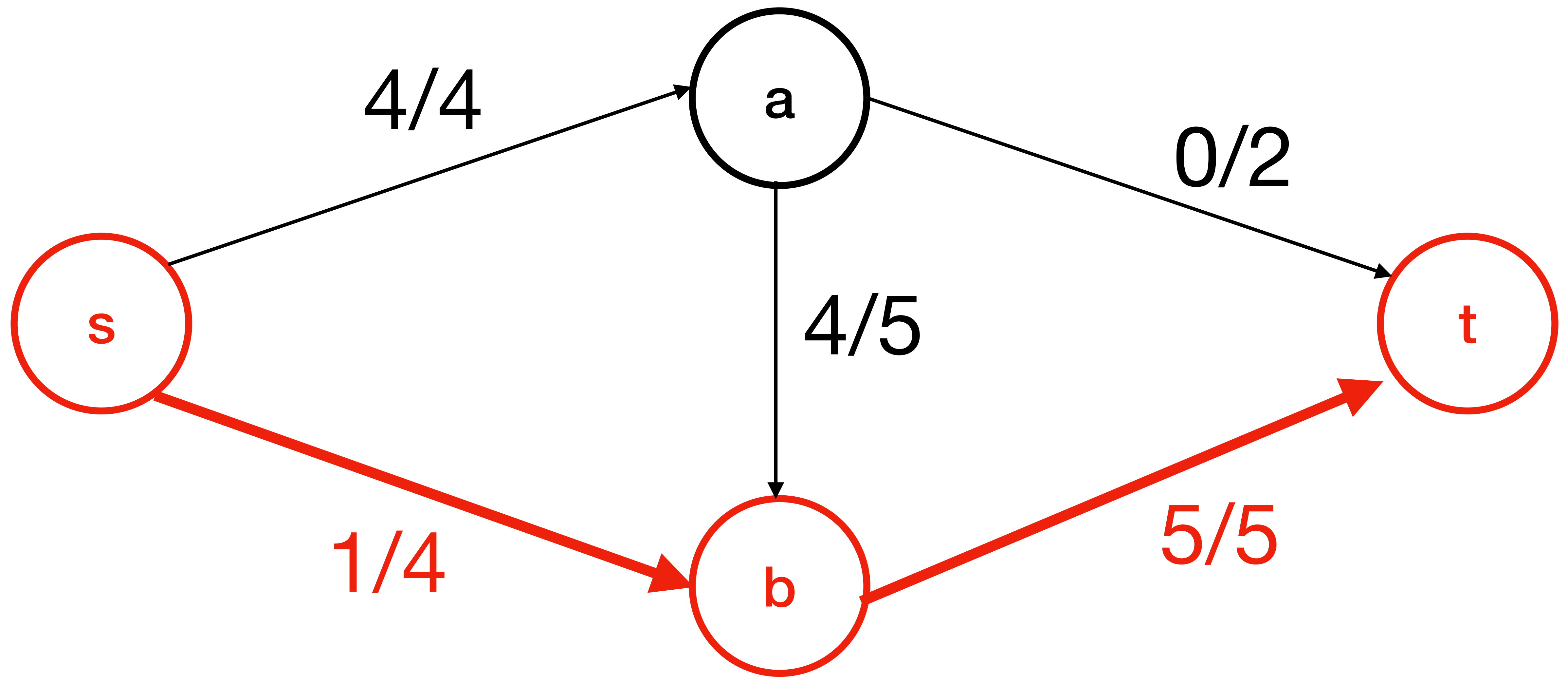


$$\delta = \min\{4,\ 1\}$$

Augmenting Path

- A path from s to t such that the flow can be increased

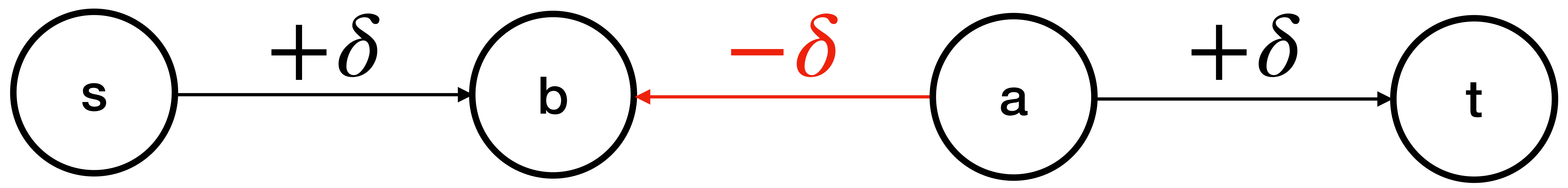


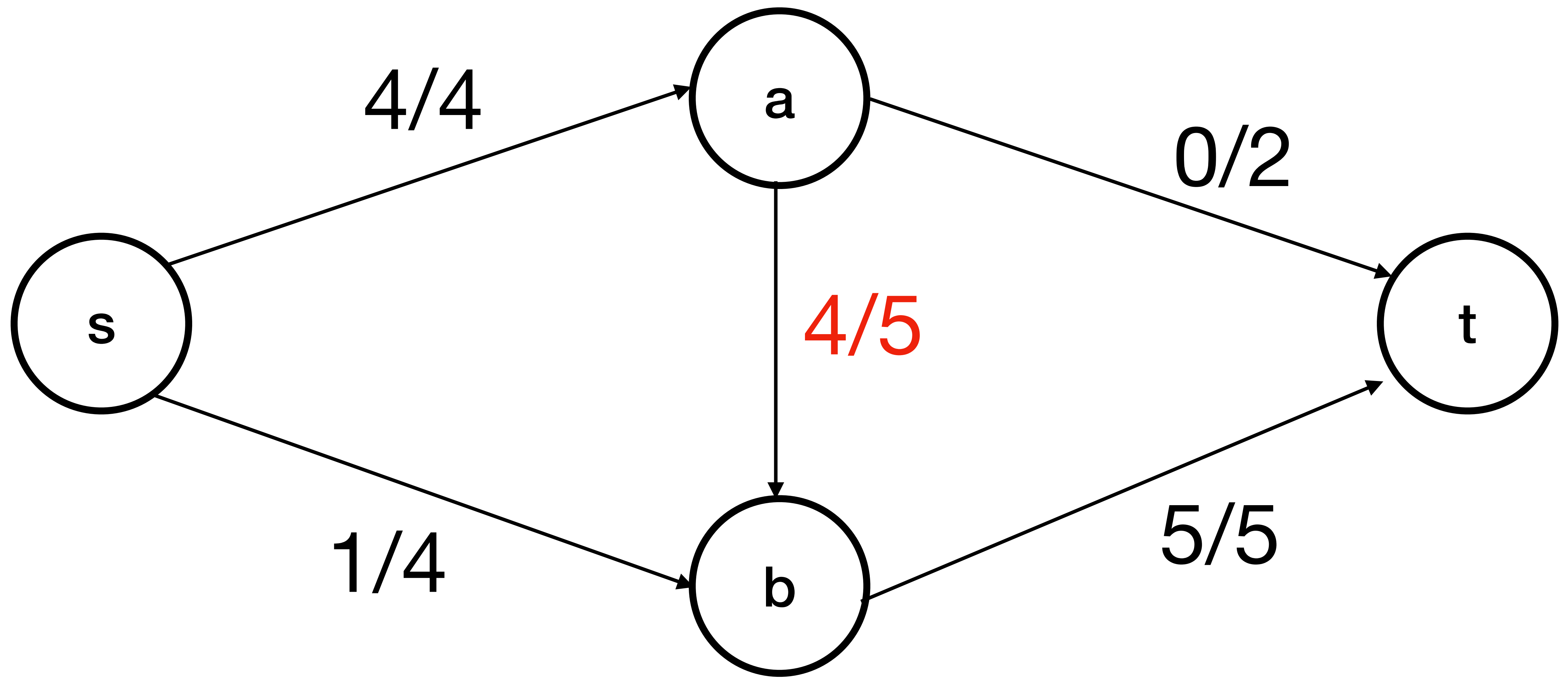


$$\delta = \min\{4,\ 1\}$$

Augmenting Path

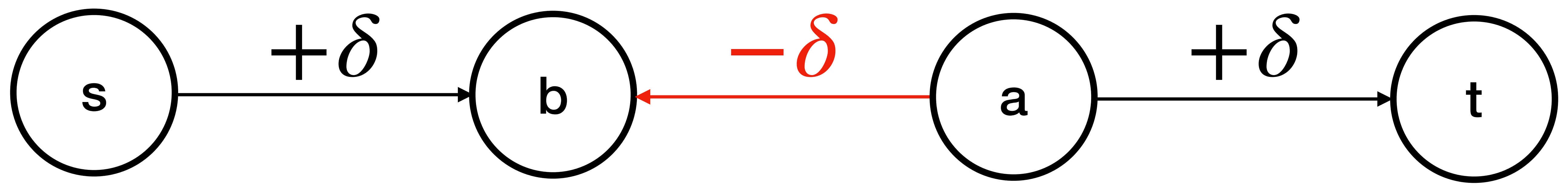
- A path from s to t such that the flow can be increased

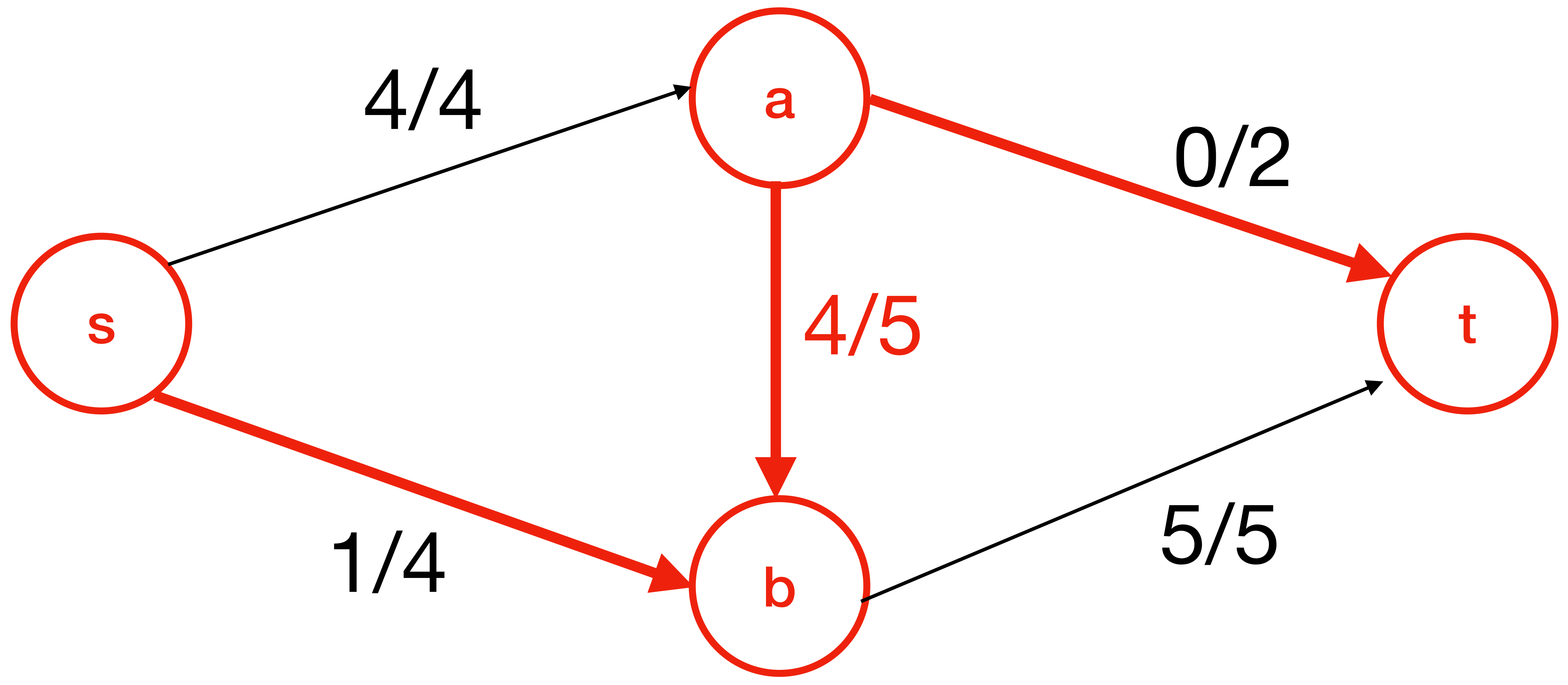




Augmenting Path

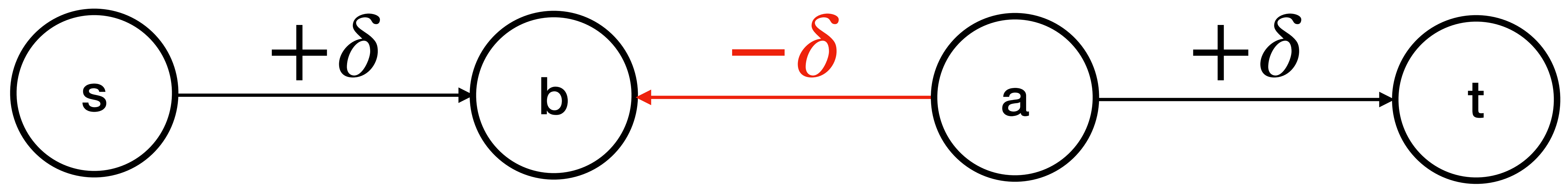
- A path from s to t such that the flow can be increased

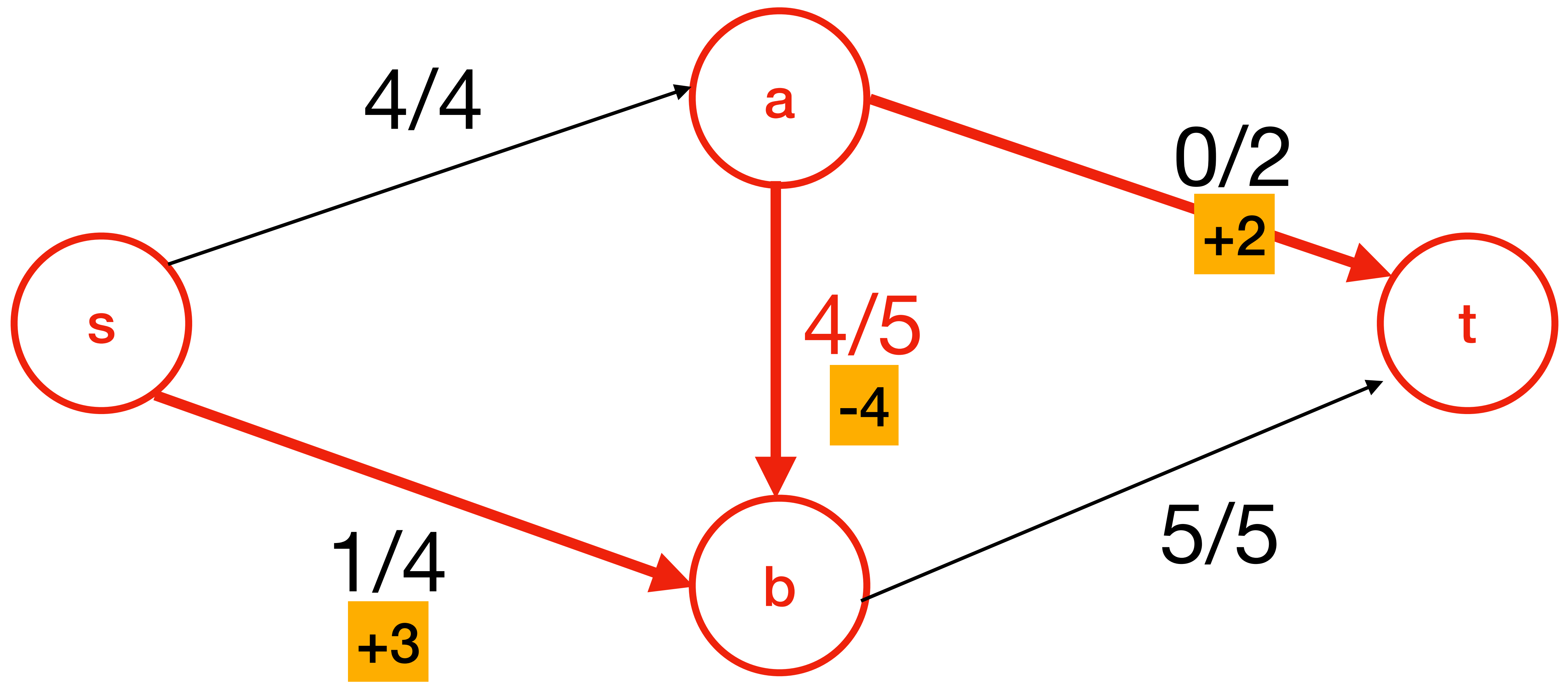




Augmenting Path

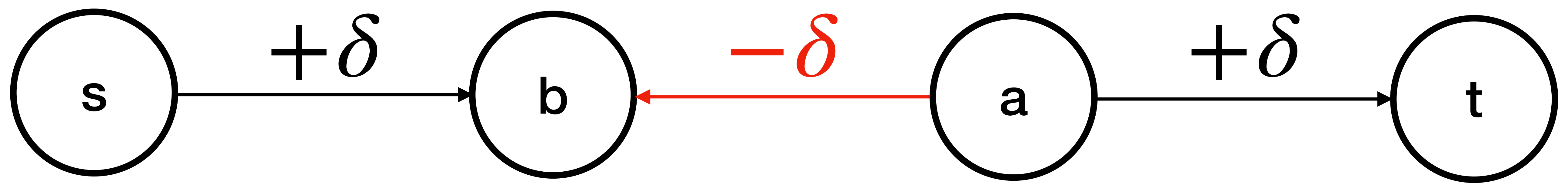
- A path from s to t such that the flow can be increased

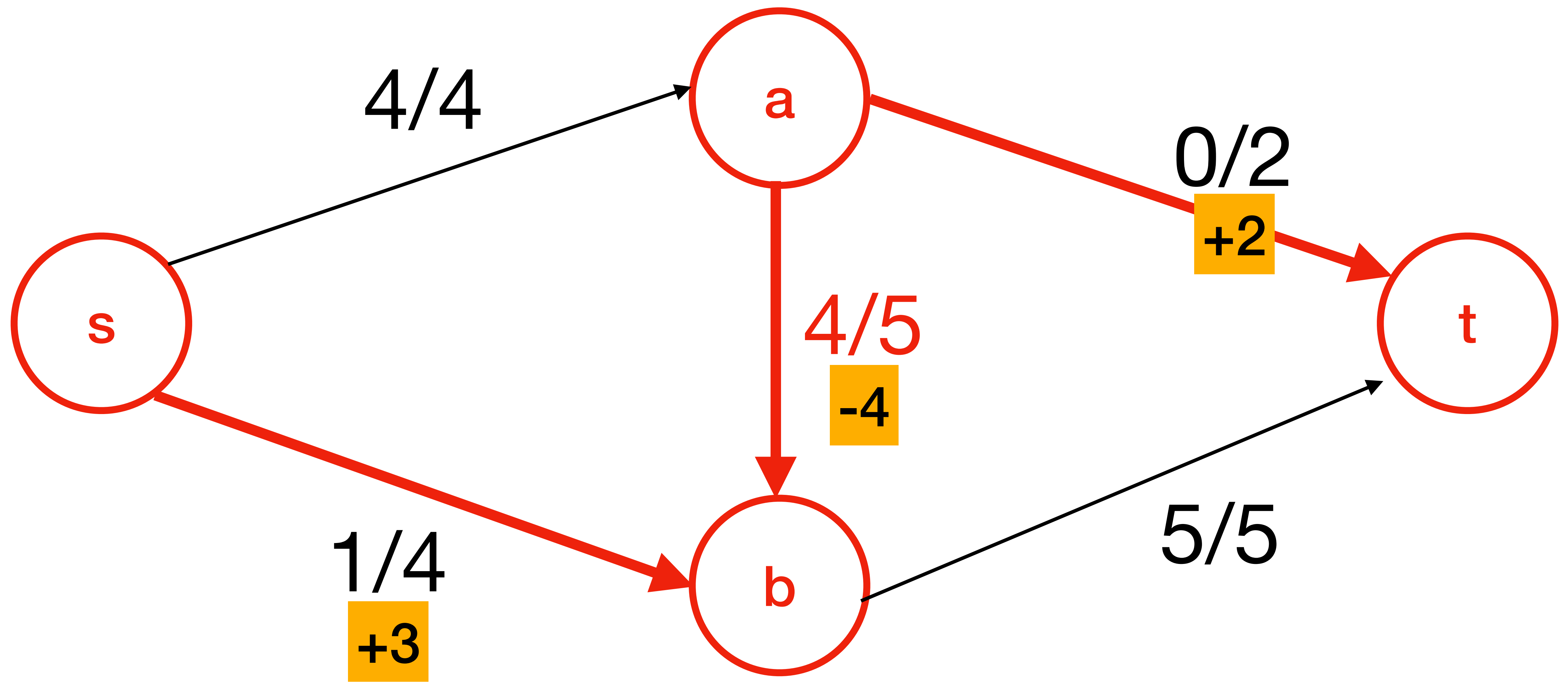




Augmenting Path

- A path from s to t such that the flow can be increased

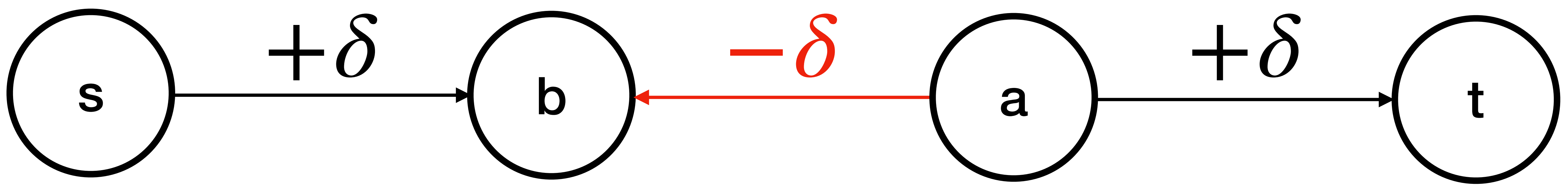


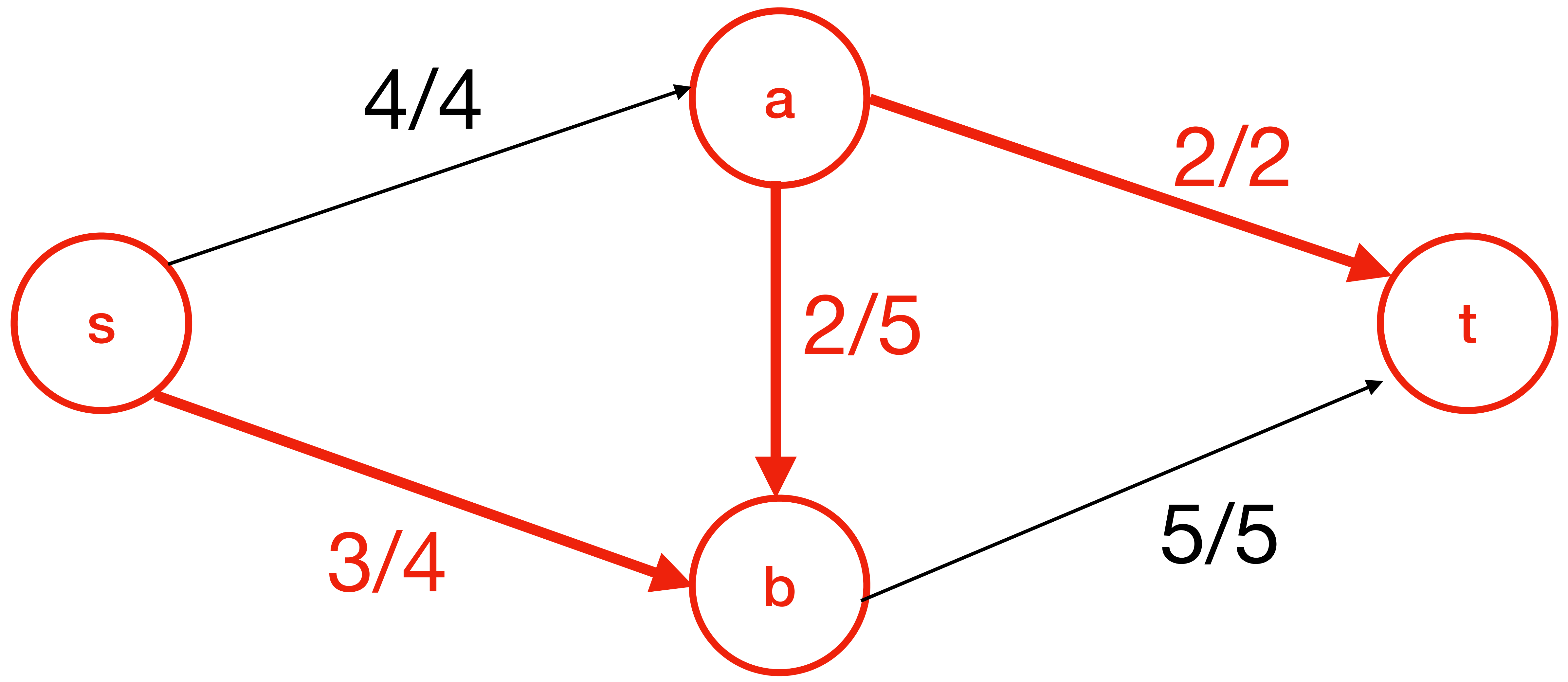


$$\delta = \min\{3,\ 4,\ 2\}$$

Augmenting Path

- A path from s to t such that the flow can be increased





$$\delta = \min\{3,\ 4,\ 2\}$$

No more augmenting path

Augmenting Path (Summary)

- A path from s to t such that the flow can be increased

$$\begin{tikzcd} s \arrow[r, "+\delta"] & a \arrow[r, "+\delta"] & b \arrow[r, "+\delta"] & t \end{tikzcd}$$

- need to decrease flow for reversed arc

$$\begin{tikzcd} s \arrow[r, "+\delta"] & b & a \arrow[l, "-\delta"'] \arrow[r, "+\delta"] & t \end{tikzcd}$$

Ford-Fulkerson Algorithm

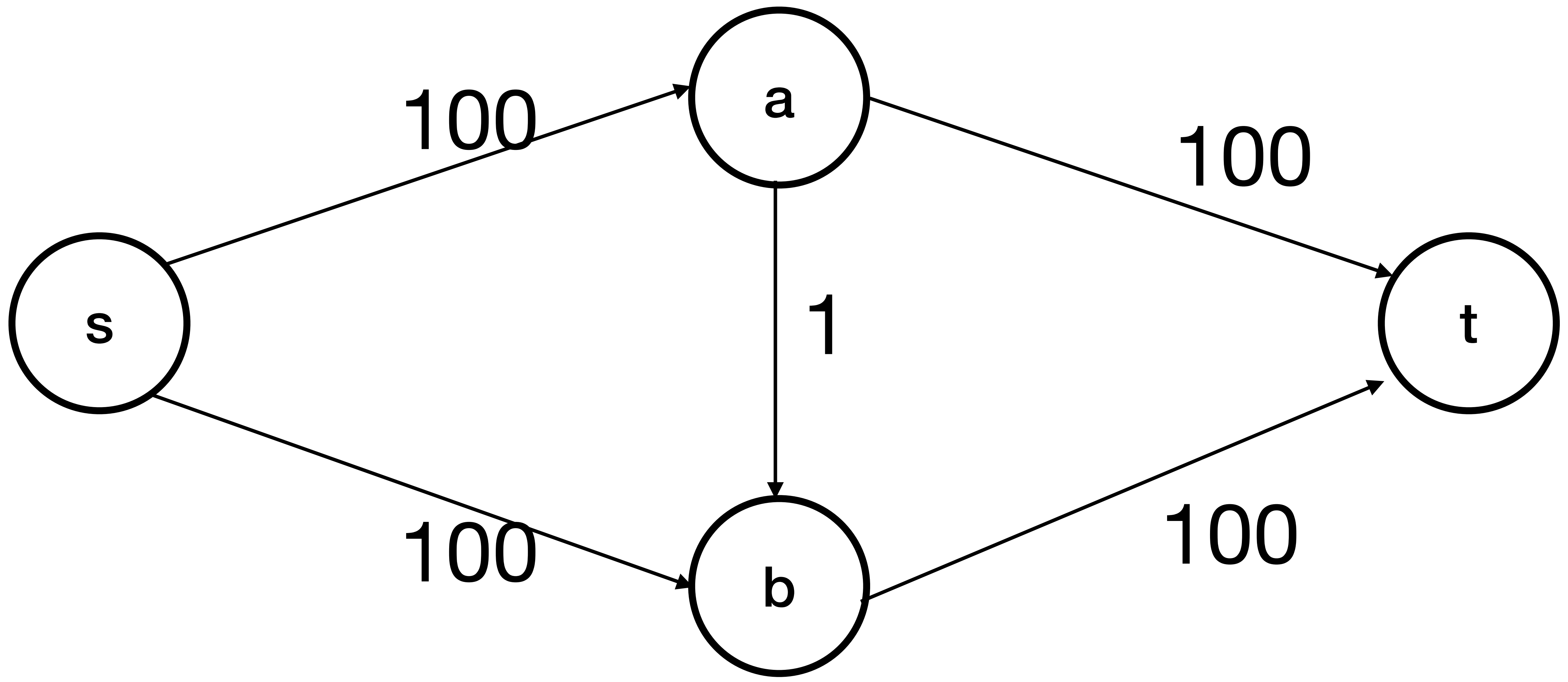
```
$f(u,v) \leftarrow 0$ for all edges $(u,v)$
while there is a path $p$ from $s$ to $t$ in $G_f$ such that $c_f(u,v) > 0 \ \forall (u,v) \in p$
    calculate $c_f(p) = \min\{c_f(u,v) : (u,v) \in p\}$
    for each edge $(u,v) \in p$
        $f(u,v) \leftarrow f(u,v) + c_f(p)$ //Send flow along the path
        $f(v,u) \leftarrow f(v,u) - c_f(p)$ //Remove flow on reversed arc
    end for
end while
```

Ford-Fulkerson Algorithm

Worst case running time

Augmenting paths

- s-a-b-t
- s-b-a-t



$O(E|f_{\max}|)$

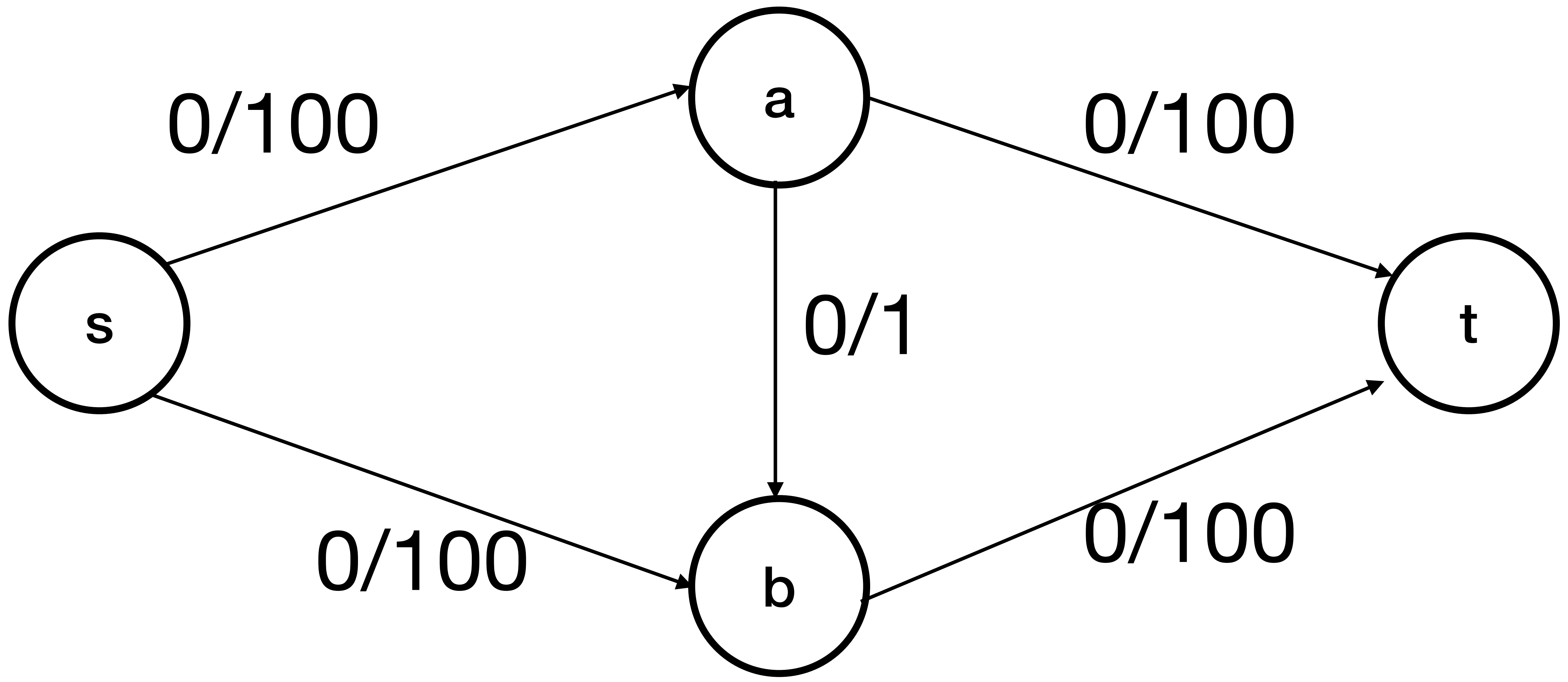
Summary

- Ford-Fulkerson $O(E|f_{\max}|)$
- Edmonds-Karp
    - Same as Ford-Fulkerson algorithm
    - Search augmenting paths with Breadth-First Search

Edmonds-Karp Algorithm

Breadth-First Search to find shortest path

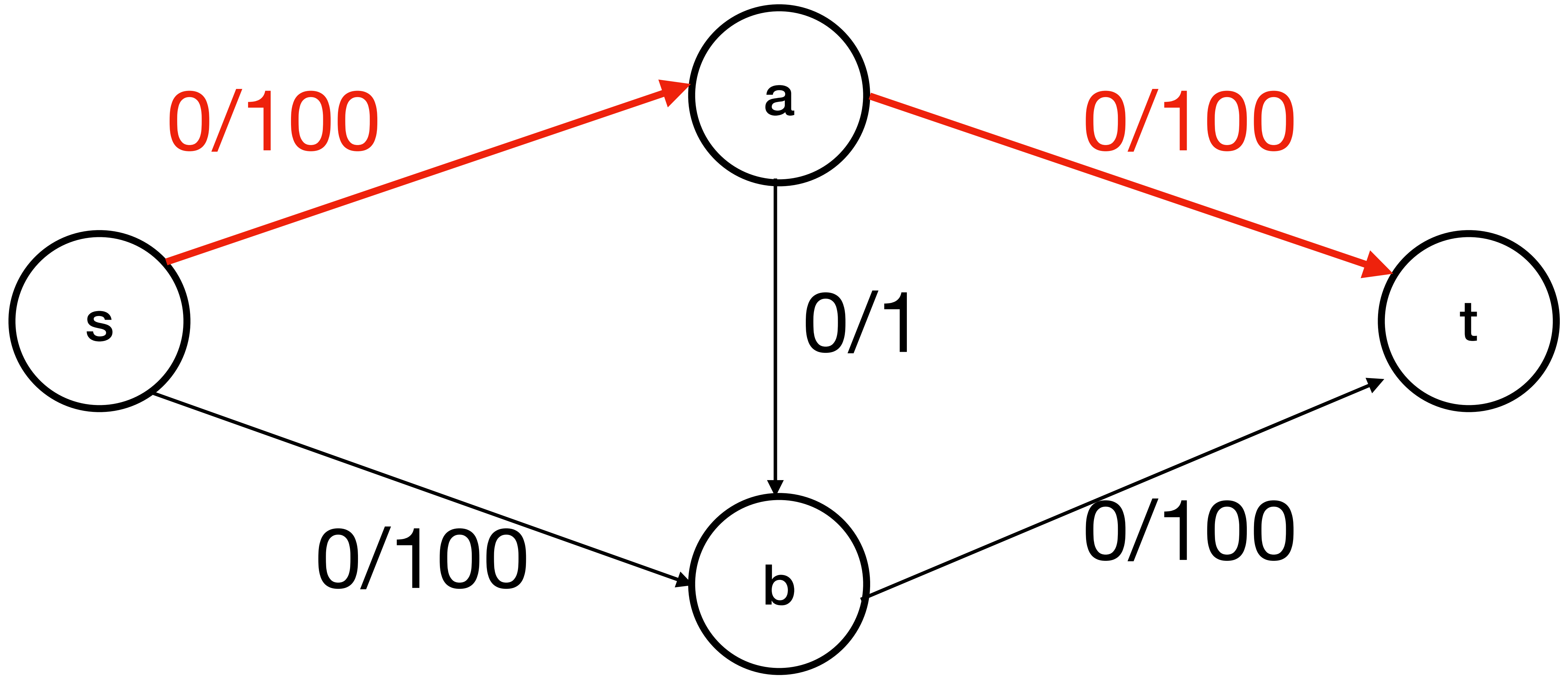
- s-a-t
- s-b-t
- s-a-b-t
- s-b-a-t



Edmonds-Karp Algorithm

Breadth-First Search to find shortest path

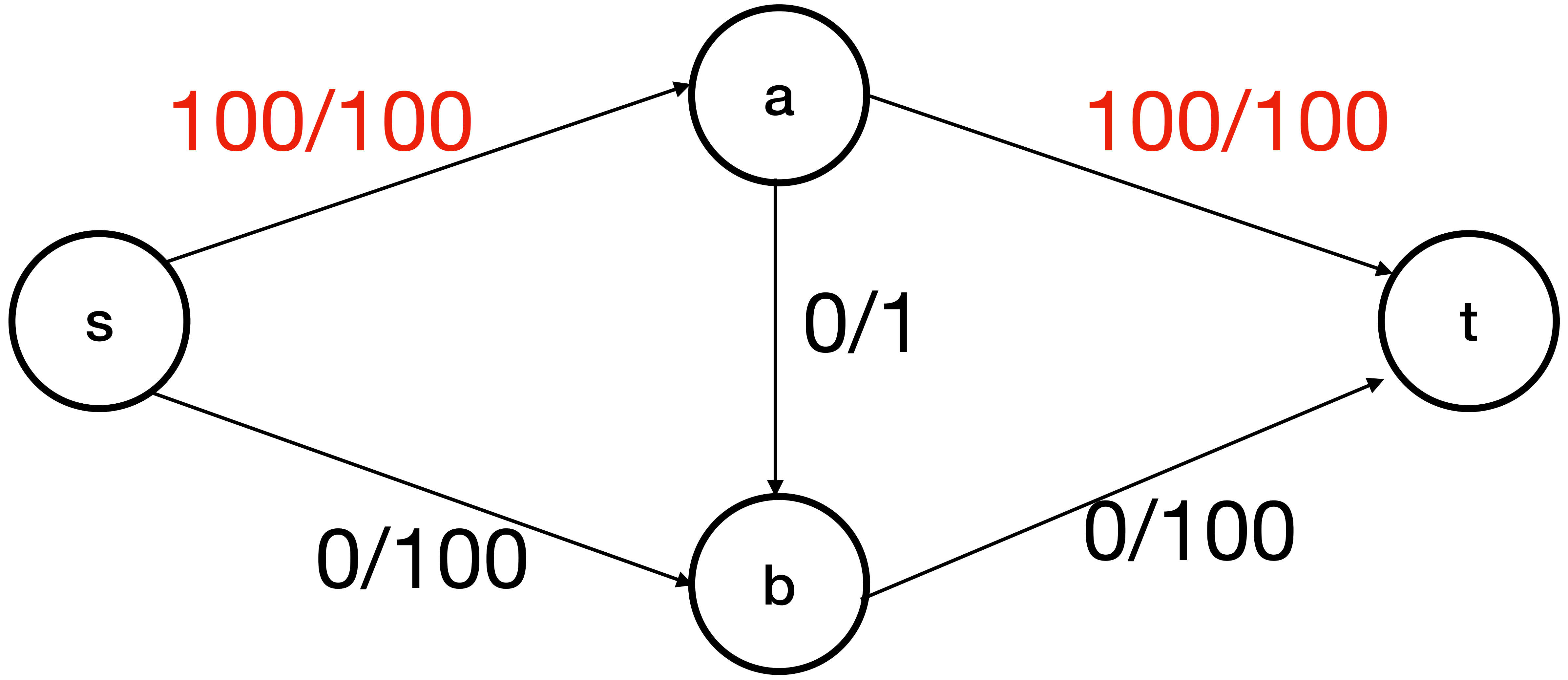
- s-a-t
- s-b-t
- s-a-b-t
- s-b-a-t



Edmonds-Karp Algorithm

Breadth-First Search to find shortest path

- s-a-t
- s-b-t
- s-a-b-t
- s-b-a-t

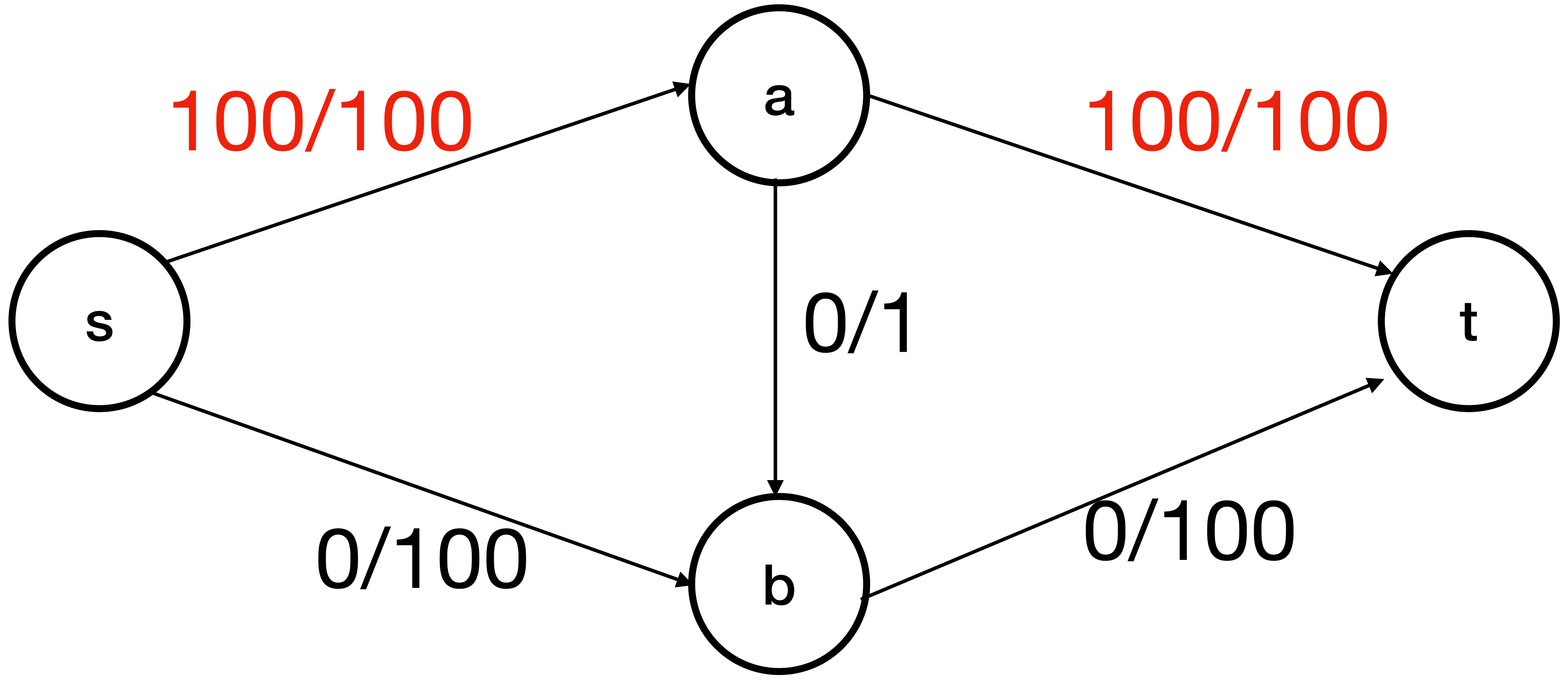


Edmonds-Karp Algorithm

Breadth-First Search to find shortest path

- ~~s-a-t~~
- s-b-t
- ~~s-a-b-t~~
- ~~s-b-a-t~~

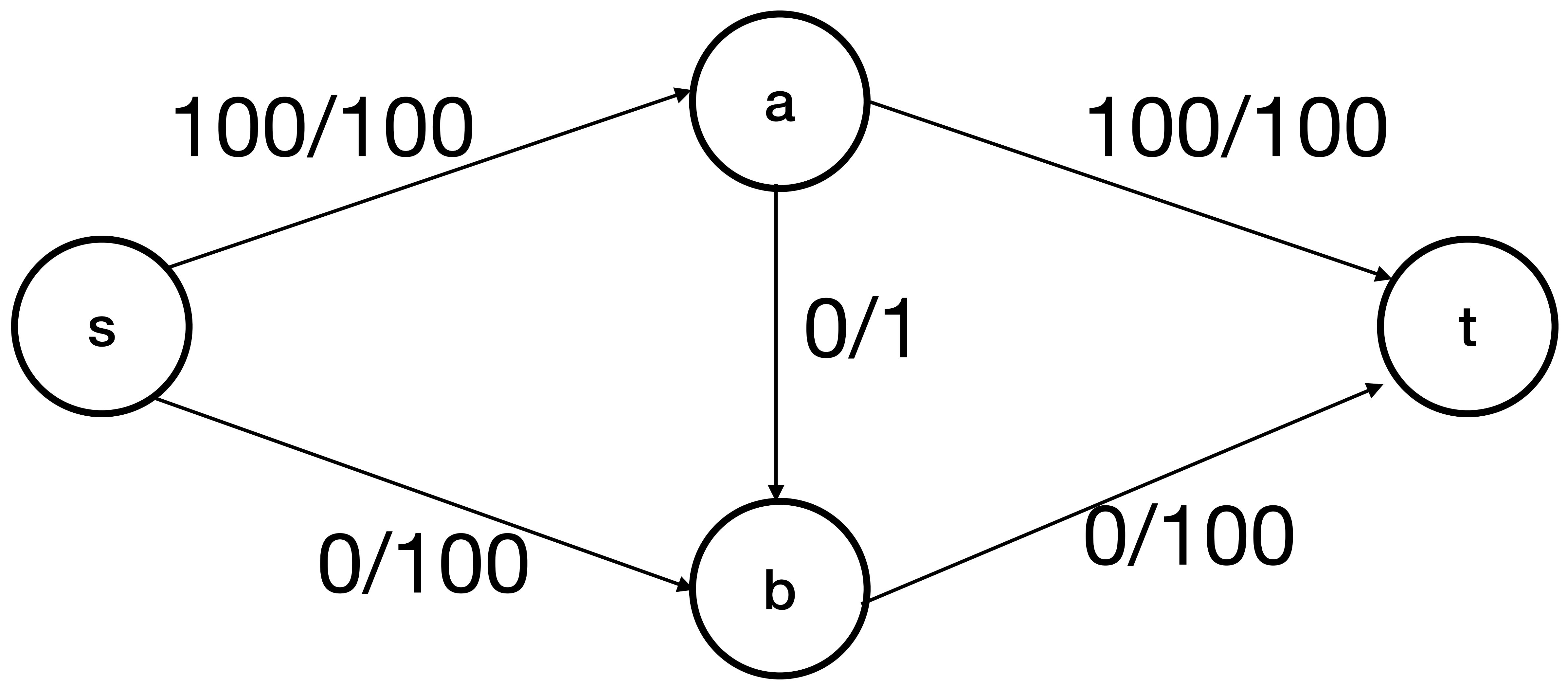
edges s->a and a->t are full, so these paths are not valid



Edmonds-Karp Algorithm

Breadth-First Search to find shortest path

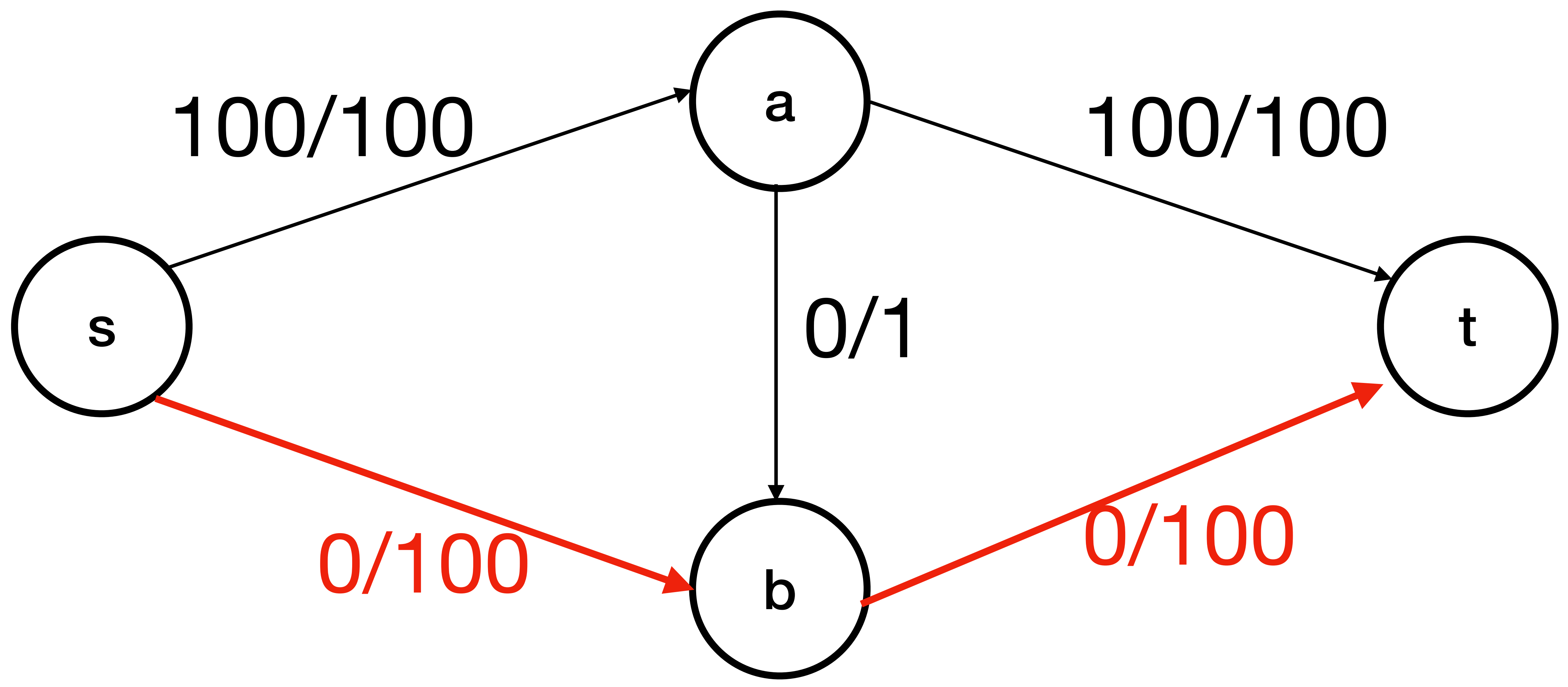
- ~~s-a-t~~
- s-b-t
- ~~s-a-b-t~~
- ~~s-b-a-t~~



Edmonds-Karp Algorithm

Breadth-First Search to find shortest path

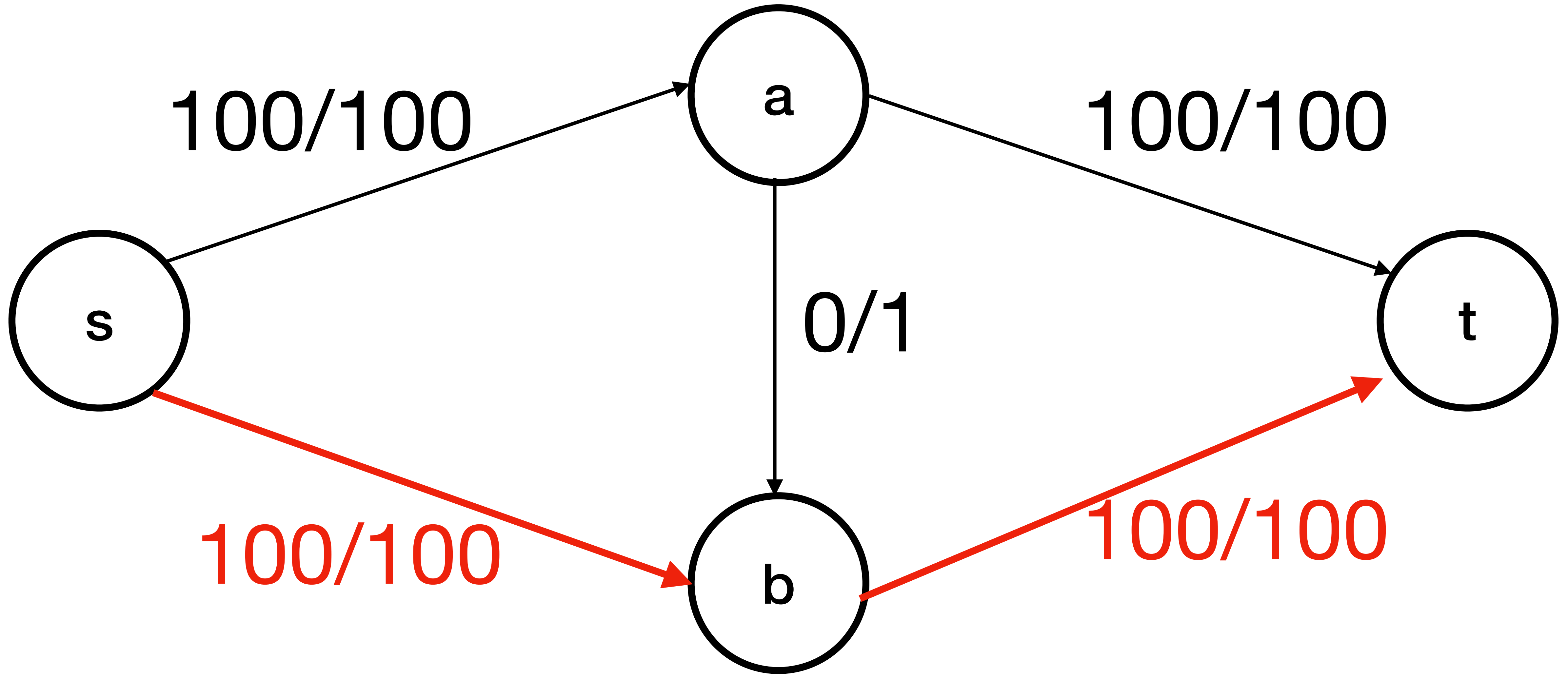
- ~~s-a-t~~
- s-b-t
- ~~s-a-b-t~~
- ~~s-b-a-t~~



Edmonds-Karp Algorithm

Breadth-First Search to find shortest path

- ~~s-a-t~~
- s-b-t
- ~~s-a-b-t~~
- ~~s-b-a-t~~

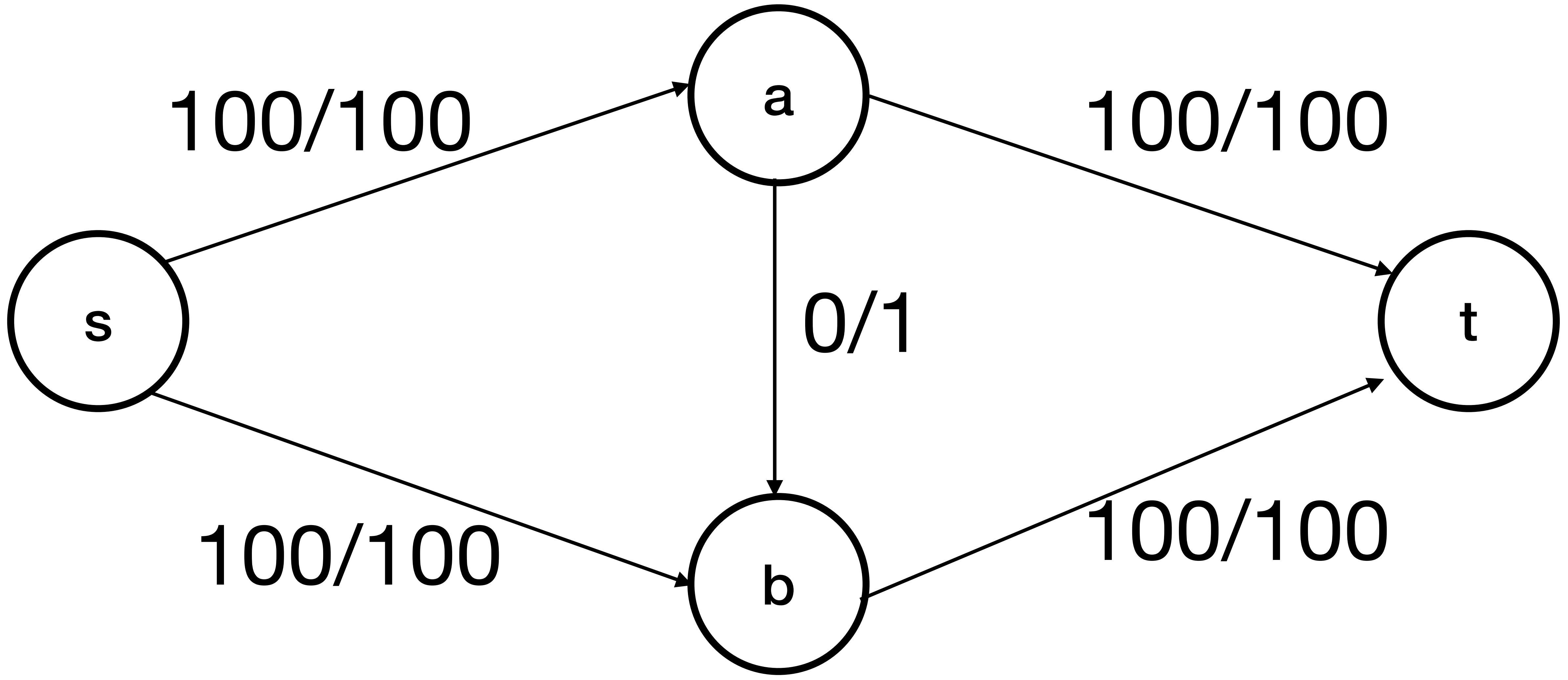


Edmonds-Karp Algorithm

Breadth-First Search to find shortest path

- ~~s-a-t~~
- ~~s-b-t~~
- ~~s-a-b-t~~
- ~~s-b-a-t~~

edge s->b is full, so this path is not valid anymore



Summary

- Ford-Fulkerson $O(E|f_{\max}|)$ 1955
- Edmonds Karp $O(VE^2)$ 1970-1972
  - Same as Ford-Fulkerson algorithm
  - Search augmenting paths with Breadth-First Search

Further references

- Push-Relabel Algorithm $O(V^2E)$ 1988
- Best algorithm $O(VE)$
  - Orlin, James B. (2013). “Max flows in O(nm) time, or better”, STOC 2013

Huffman Coding

Huffman Coding

- Huffman coding is a form of statistical coding
- Not all characters occur with the same frequency
- Yet all characters are allocated the same amount of space
    - 1 character = 1 byte = 8 bits
- Compression without loss (e.g. zip, rar)
- Proposed by David A. Huffman in 1952: “A Method for the Construction of Minimum Redundancy Codes”
- bmp (bitmap) -> jpg : Compression with loss

Idea of the coding

- Use less bits to encode characters in general
- a character that appears often should be encoded with few bits
- on the contrary, a character that does not appear often should be encoded with more bits

Algorithm

1. Scan text to be compressed and count occurrence of all characters
2. Sort or prioritize characters based on number of occurrences in text
3. Build Huffman code tree based on prioritized list
4. Perform a traversal of tree to determine all code words
5. Scan text again and create new file using the Huffman codes

Algorithm

1. Scan text to be compressed and count occurrence of all characters
2. Sort or prioritize characters based on number of occurrences in text
3. Build Huffman code tree based on prioritized list
4. Perform a traversal of tree to determine all code words
5. Scan text again and create new file using the Huffman codes

Eerie eyes seen near lake.

E	e	r	i	y	s	n	a	l	k	“space”	.
1	8	2	1	1	2	2	2	1	1	4	1

Algorithm

1. Scan text to be compressed and count occurrence of all characters
2. Sort or prioritize characters based on number of occurrences in text
3. Build Huffman code tree based on prioritized list
4. Perform a traversal of tree to determine all code words
5. Scan text again and create new file using the Huffman codes

Eerie eyes seen near lake.

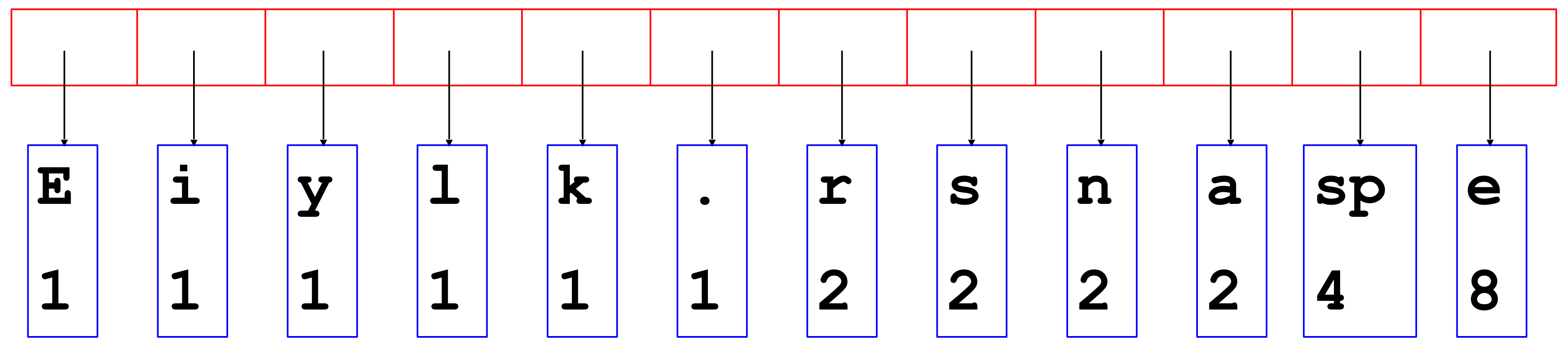
E	i	y	l	k	.	r	s	n	a	“space”	e
1	1	1	1	1	1	2	2	2	2	4	8

Algorithm

1. Scan text to be compressed and count occurrence of all characters
2. Sort or prioritize characters based on number of occurrences in text
3. Build Huffman code tree based on prioritized list
4. Perform a traversal of tree to determine all code words
5. Scan text again and create new file using the Huffman codes

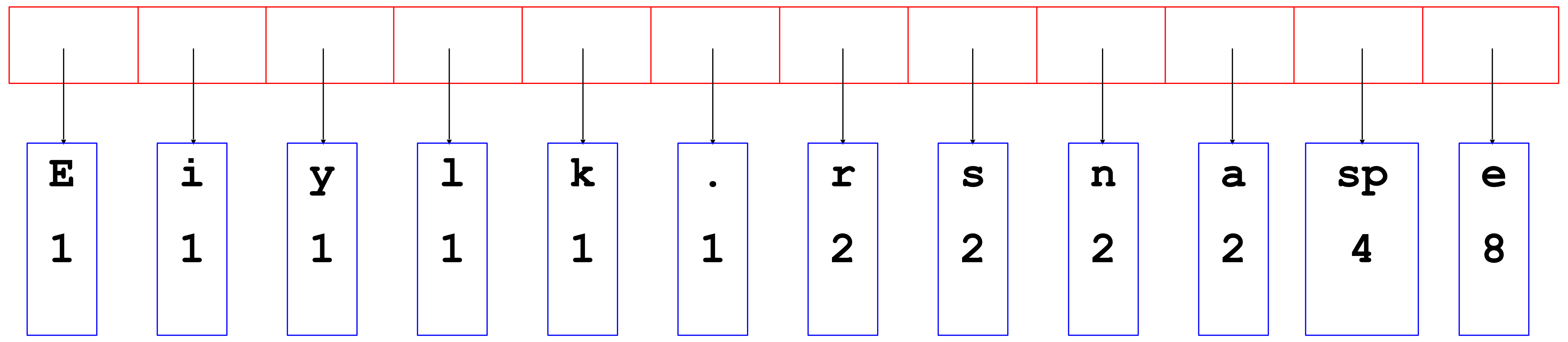
Eerie eyes seen near lake.

E	i	y	l	k	.	r	s	n	a	"space"	e
1	1	1	1	1	1	2	2	2	2	4	8



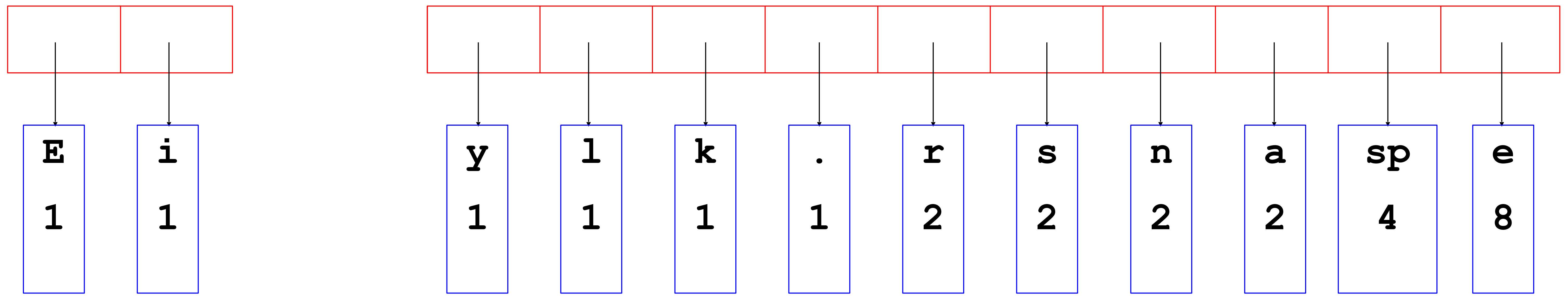
Building a tree

- While priority queue contains two or more nodes
    - Create new node
    - Dequeue node and make it left subtree
    - Dequeue next node and make it right subtree
    - Frequency of new node equals sum of frequency of left and right children
    - Enqueue new node back into queue



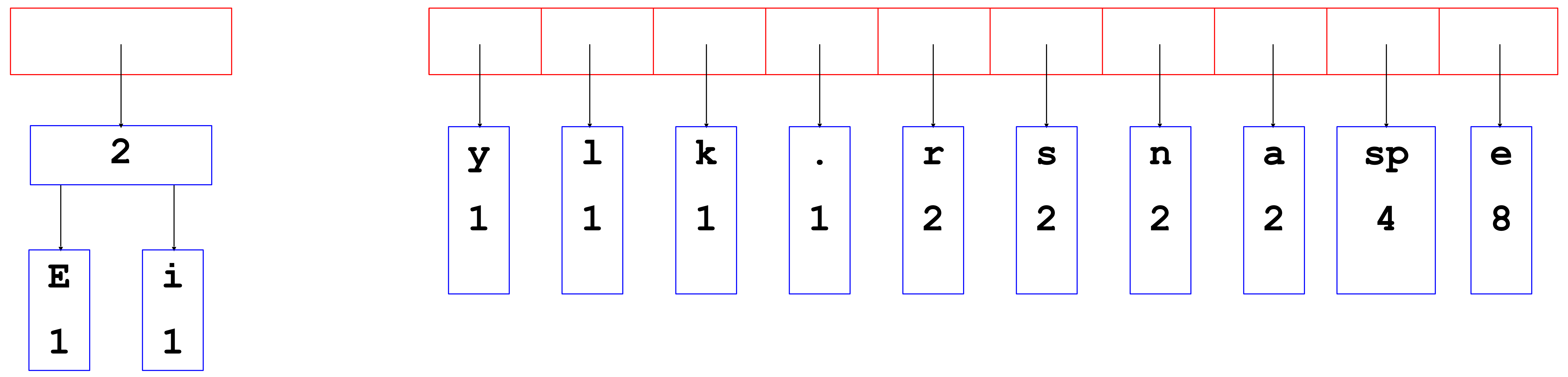
Building a tree

- While priority queue contains two or more nodes
    - Create new node
    - Dequeue node and make it left subtree
    - Dequeue next node and make it right subtree
    - Frequency of new node equals sum of frequency of left and right children
    - Enqueue new node back into queue



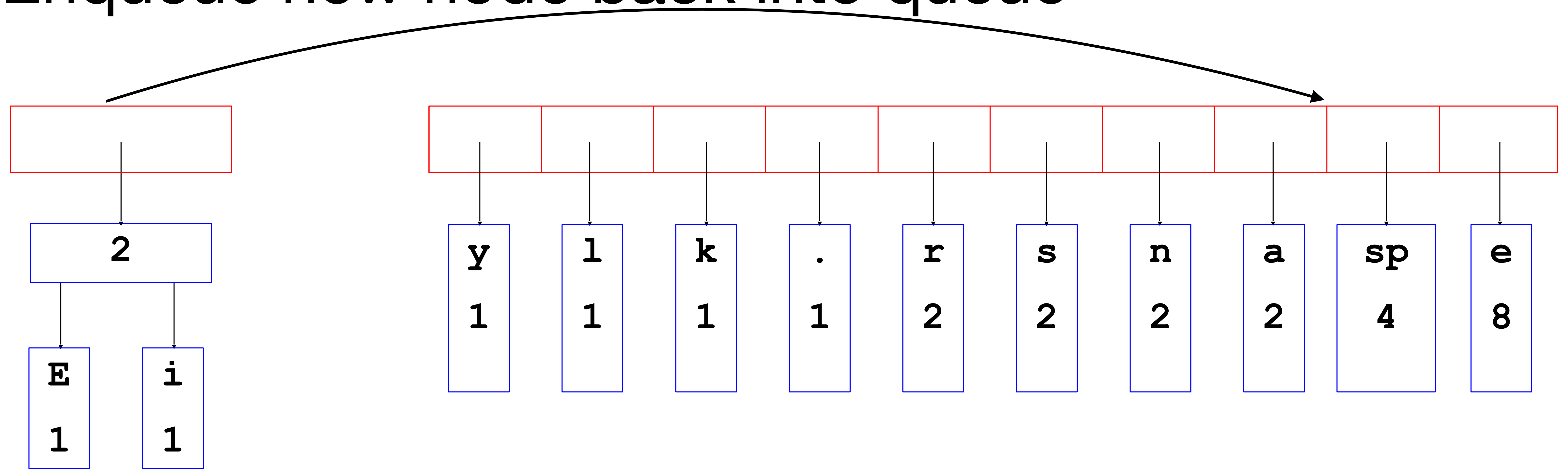
Building a tree

- While priority queue contains two or more nodes
    - Create new node
    - Dequeue node and make it left subtree
    - Dequeue next node and make it right subtree
    - Frequency of new node equals sum of frequency of left and right children
    - Enqueue new node back into queue



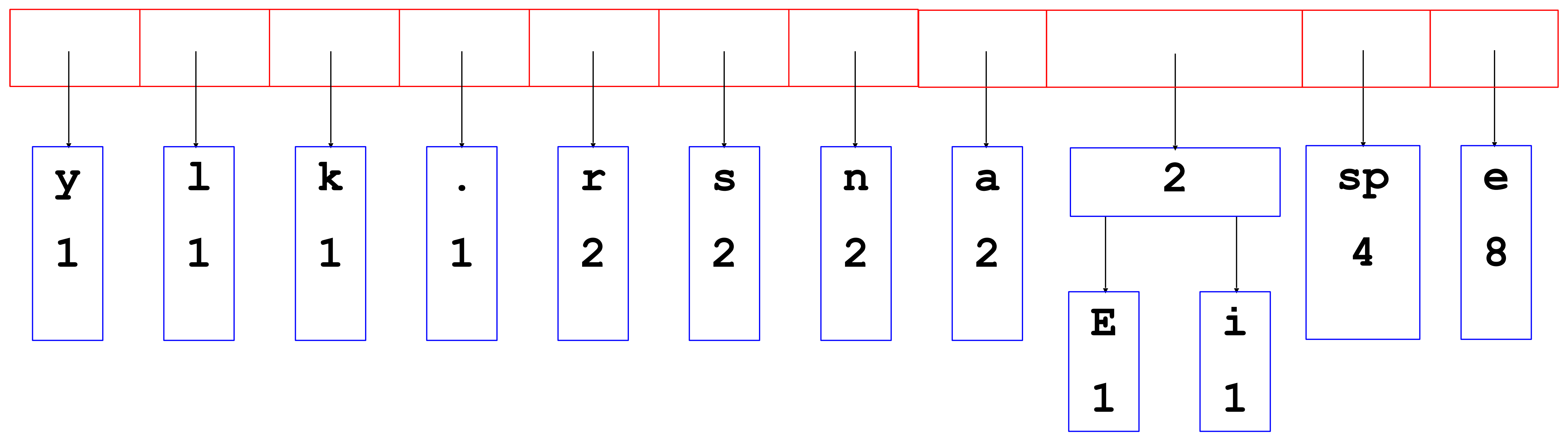
Building a tree

- While priority queue contains two or more nodes
    - Create new node
    - Dequeue node and make it left subtree
    - Dequeue next node and make it right subtree
    - Frequency of new node equals sum of frequency of left and right children
    - Enqueue new node back into queue

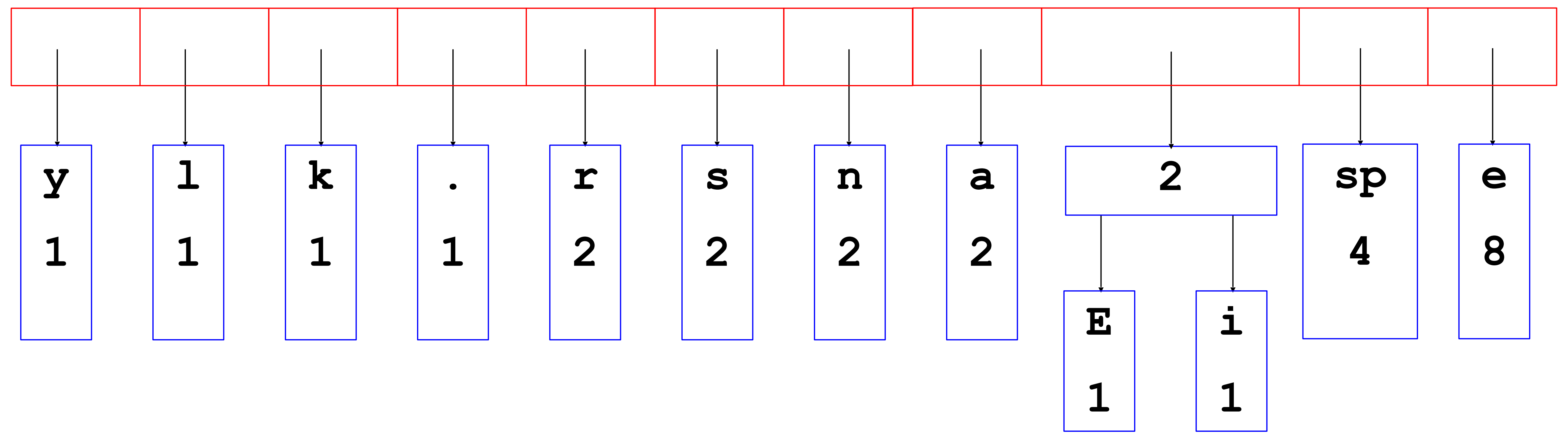


Building a tree

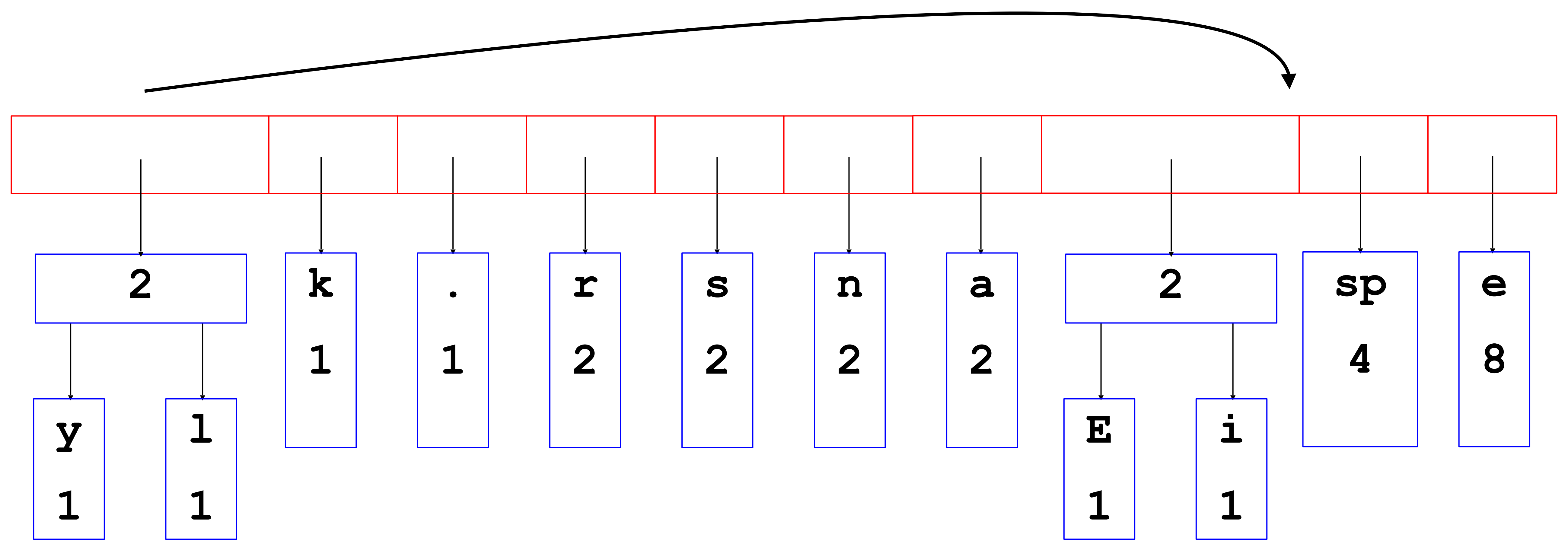
- While priority queue contains two or more nodes
    - Create new node
    - Dequeue node and make it left subtree
    - Dequeue next node and make it right subtree
    - Frequency of new node equals sum of frequency of left and right children
    - Enqueue new node back into queue



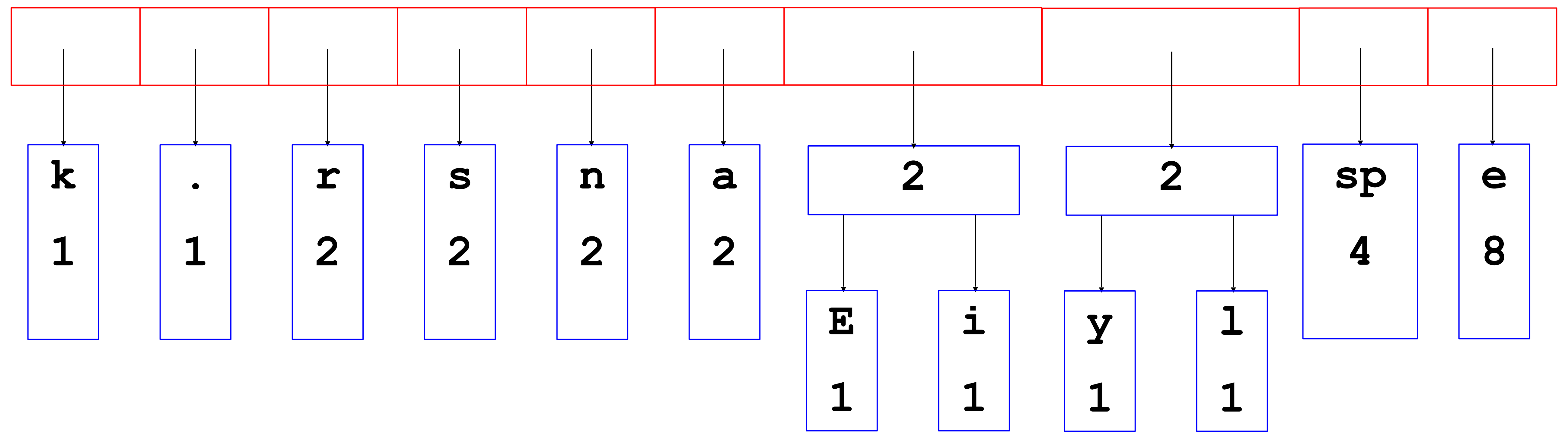
Building a tree



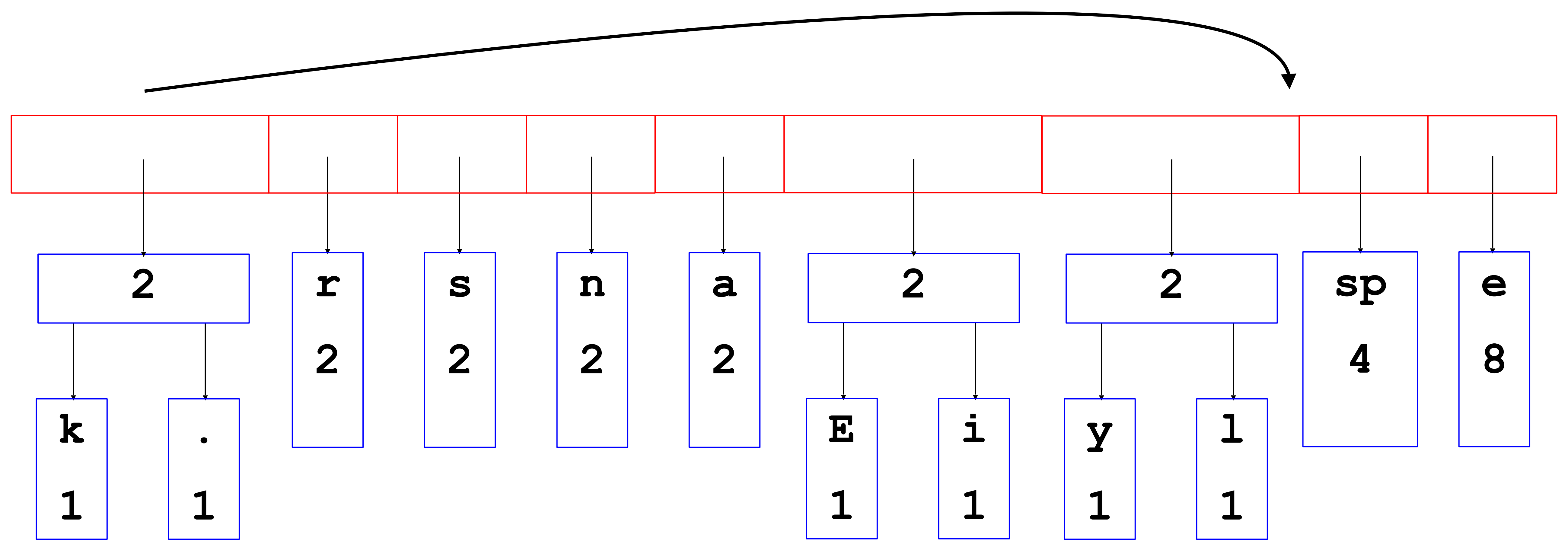
Building a tree



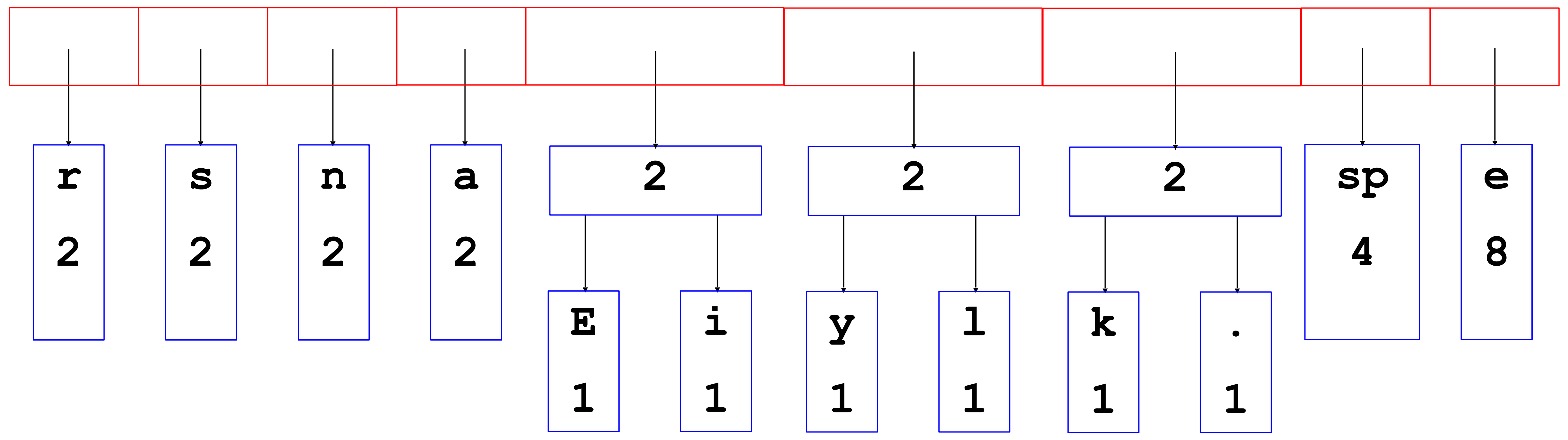
Building a tree



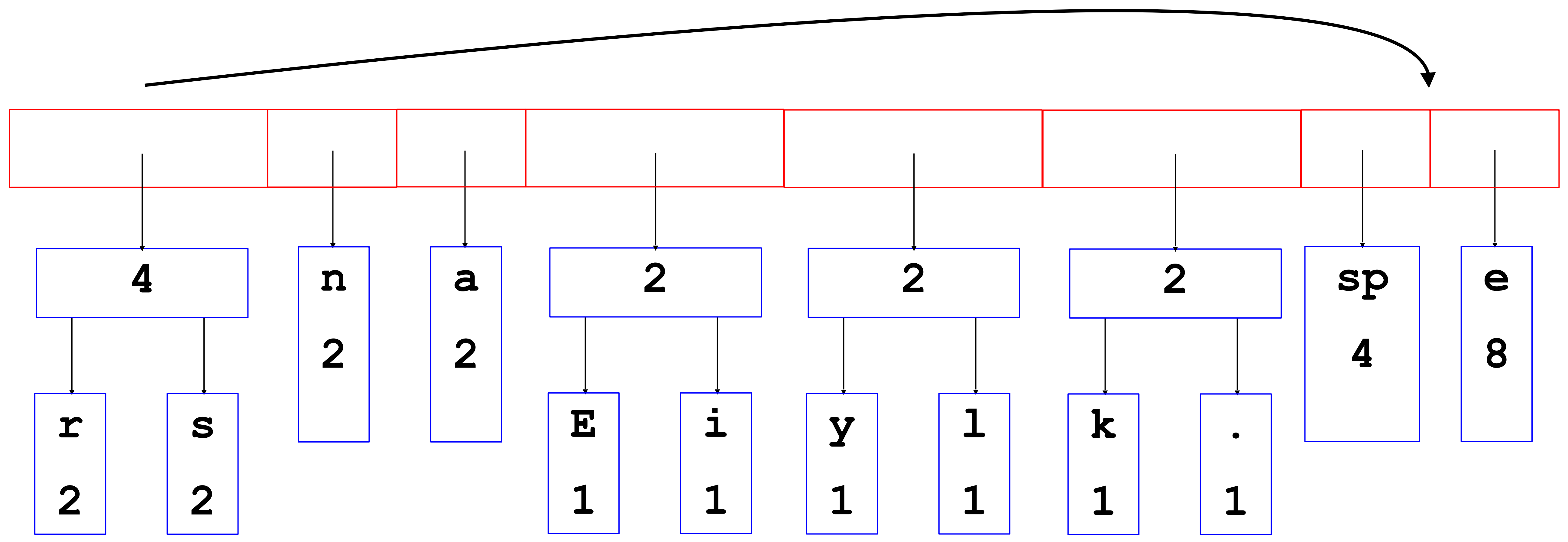
Building a tree



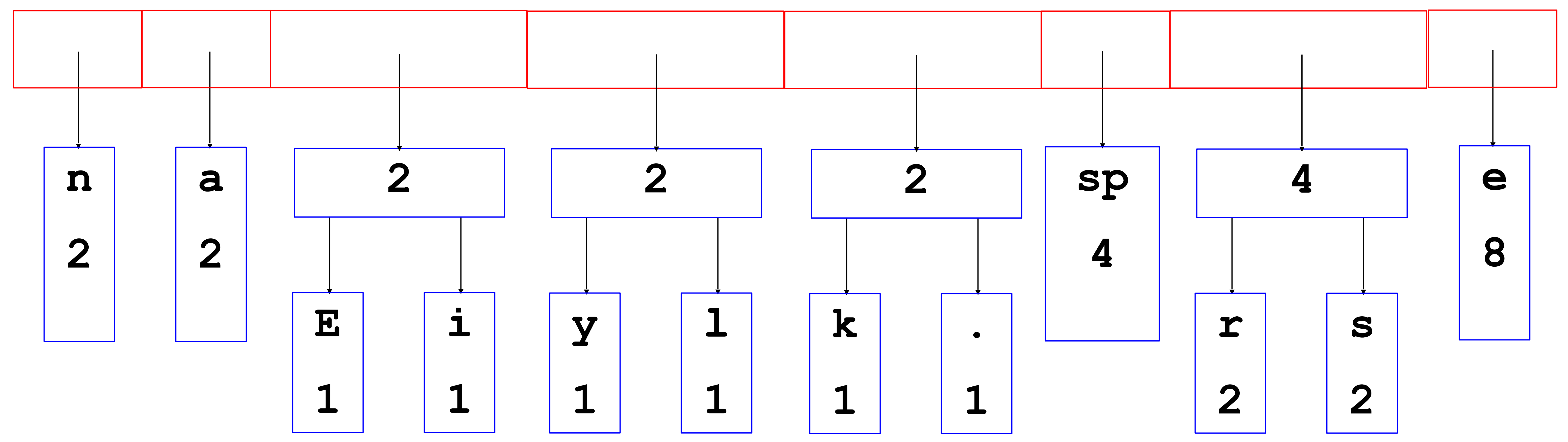
Building a tree



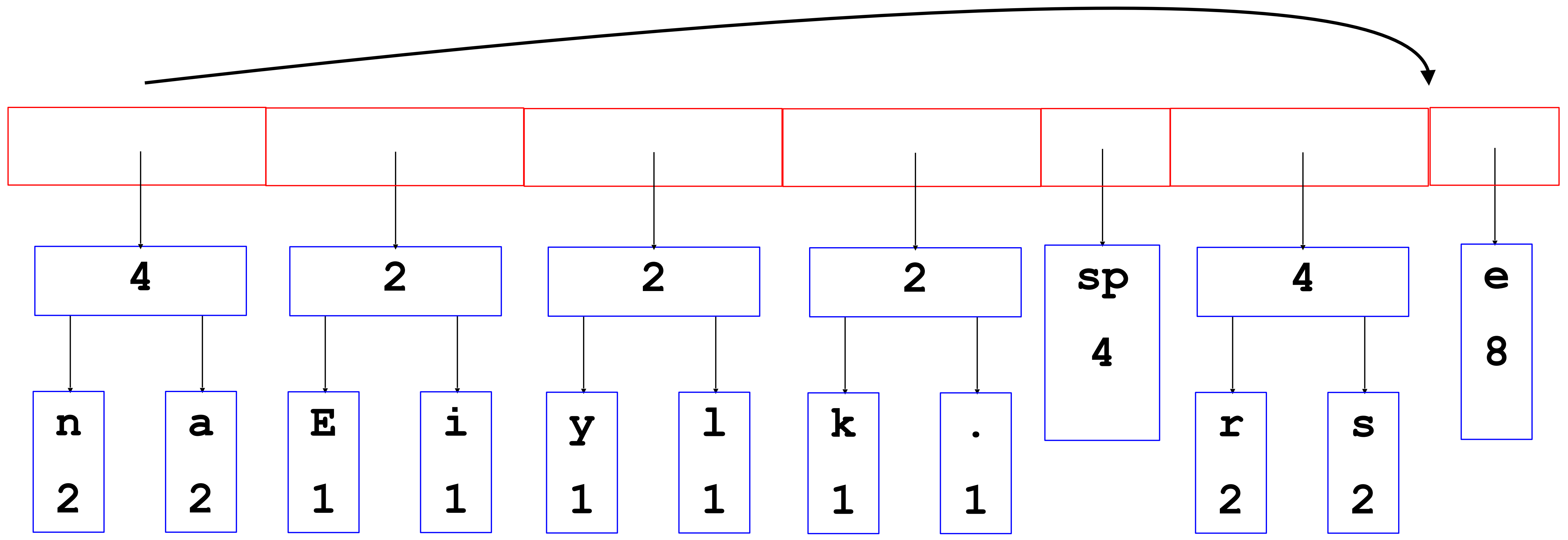
Building a tree



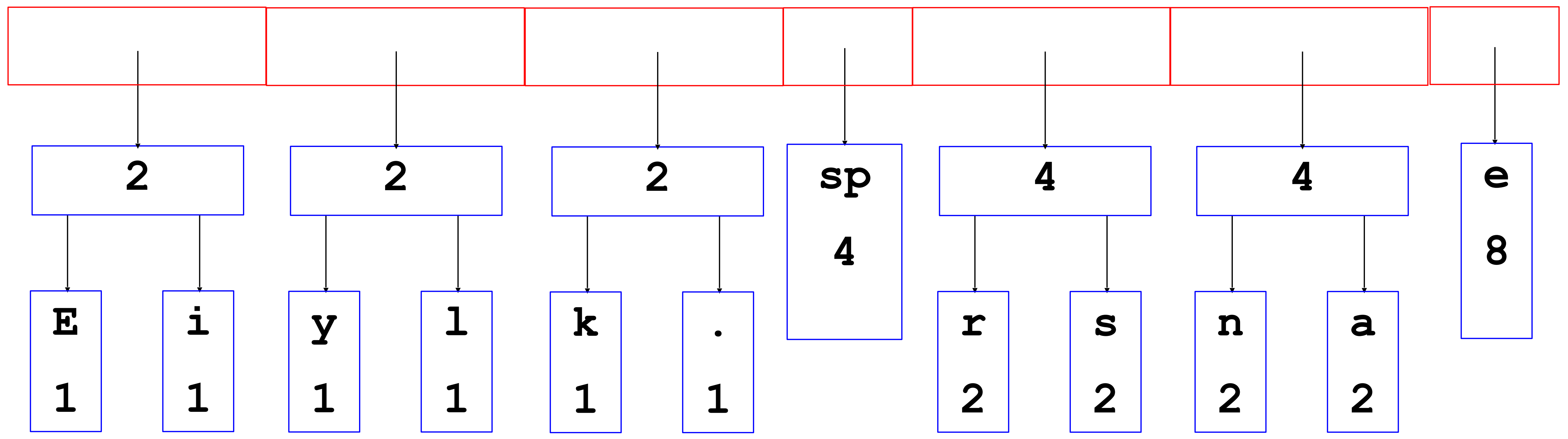
Building a tree



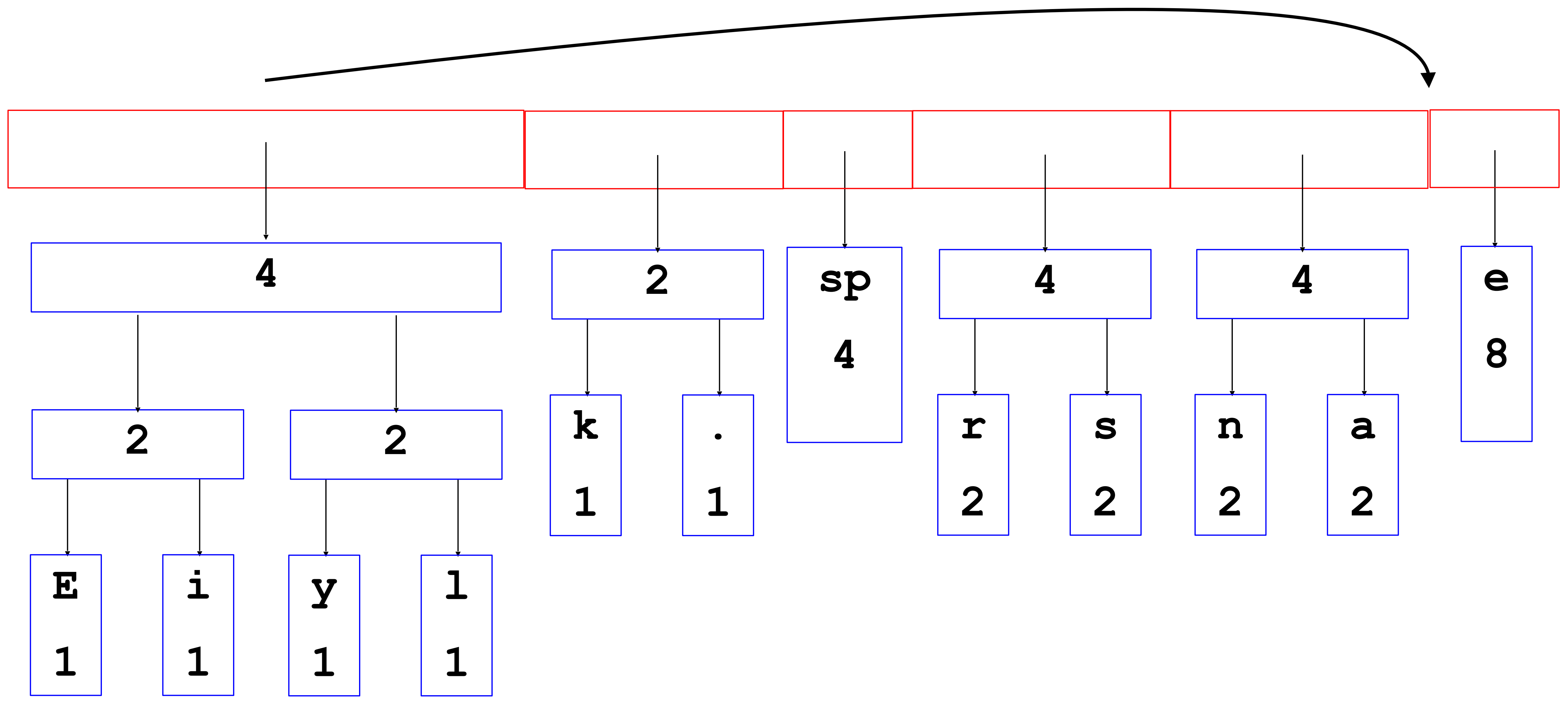
Building a tree



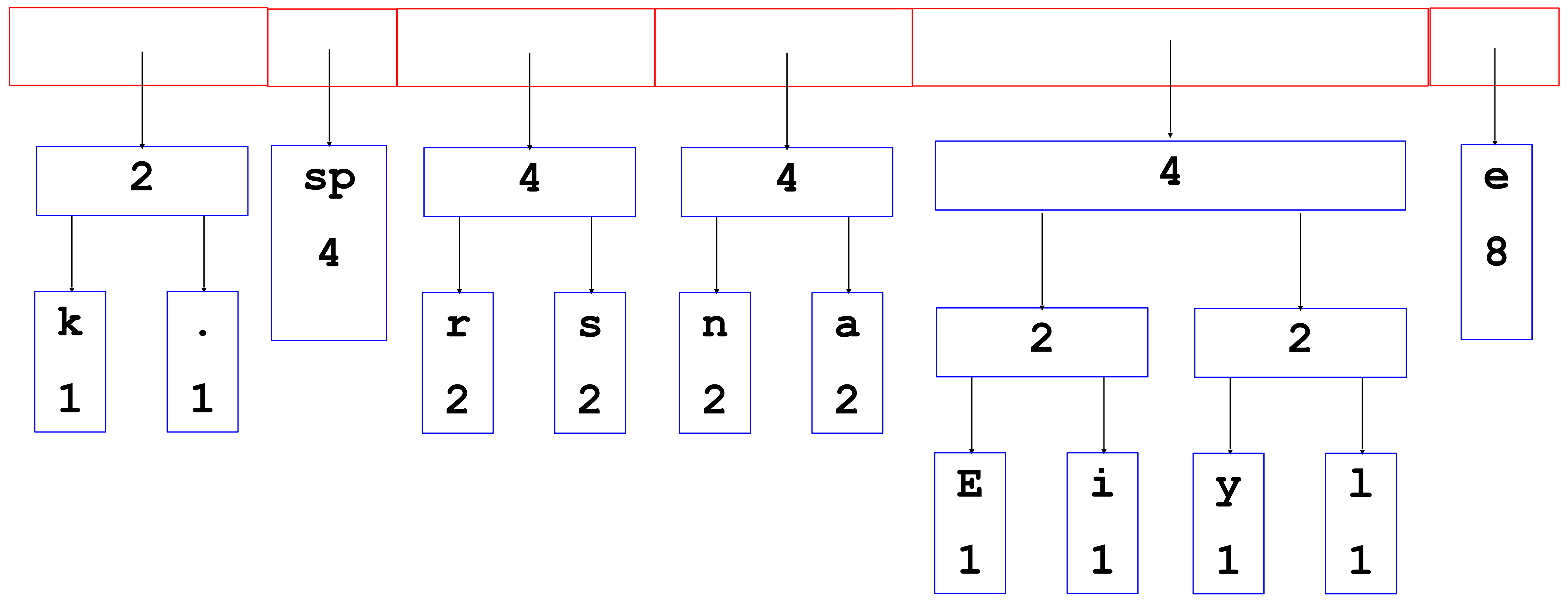
Building a tree



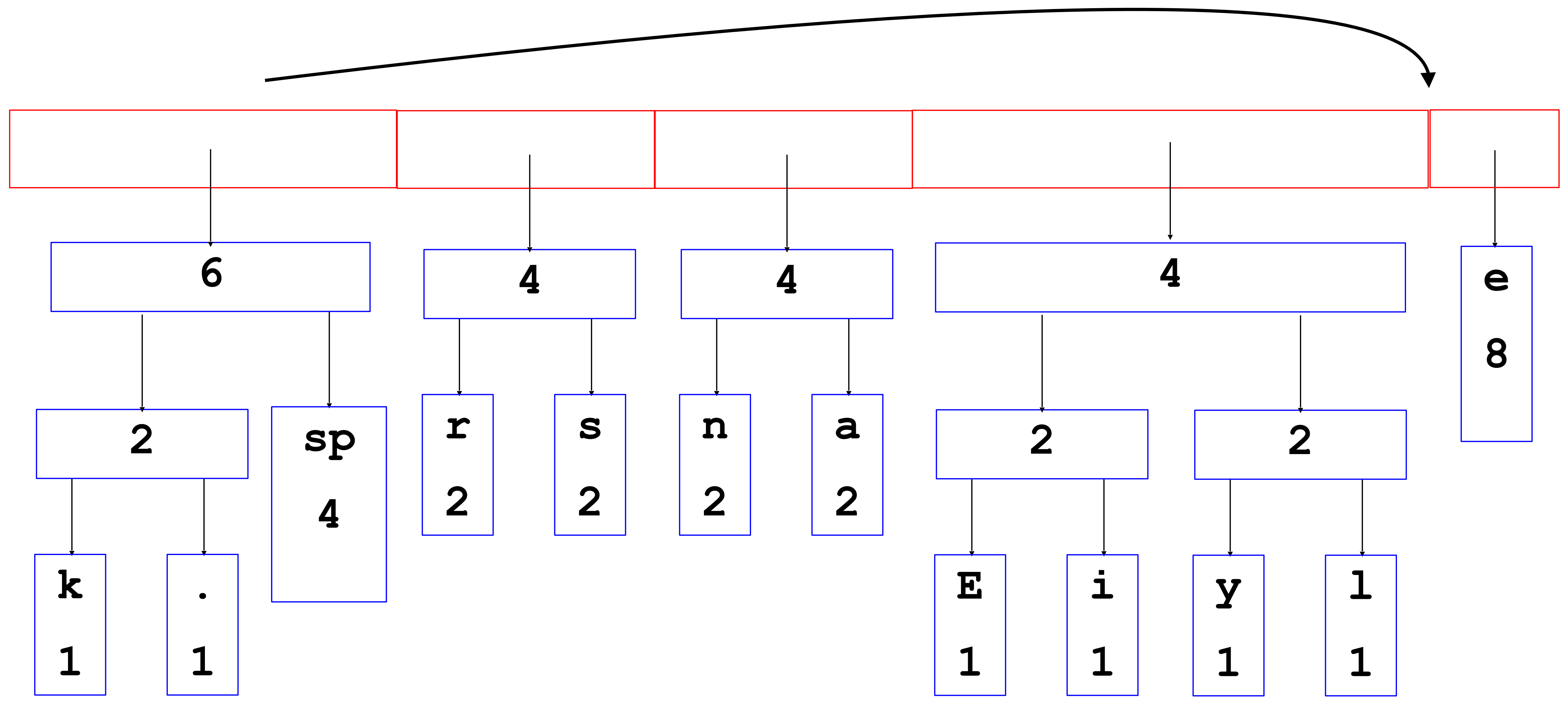
Building a tree



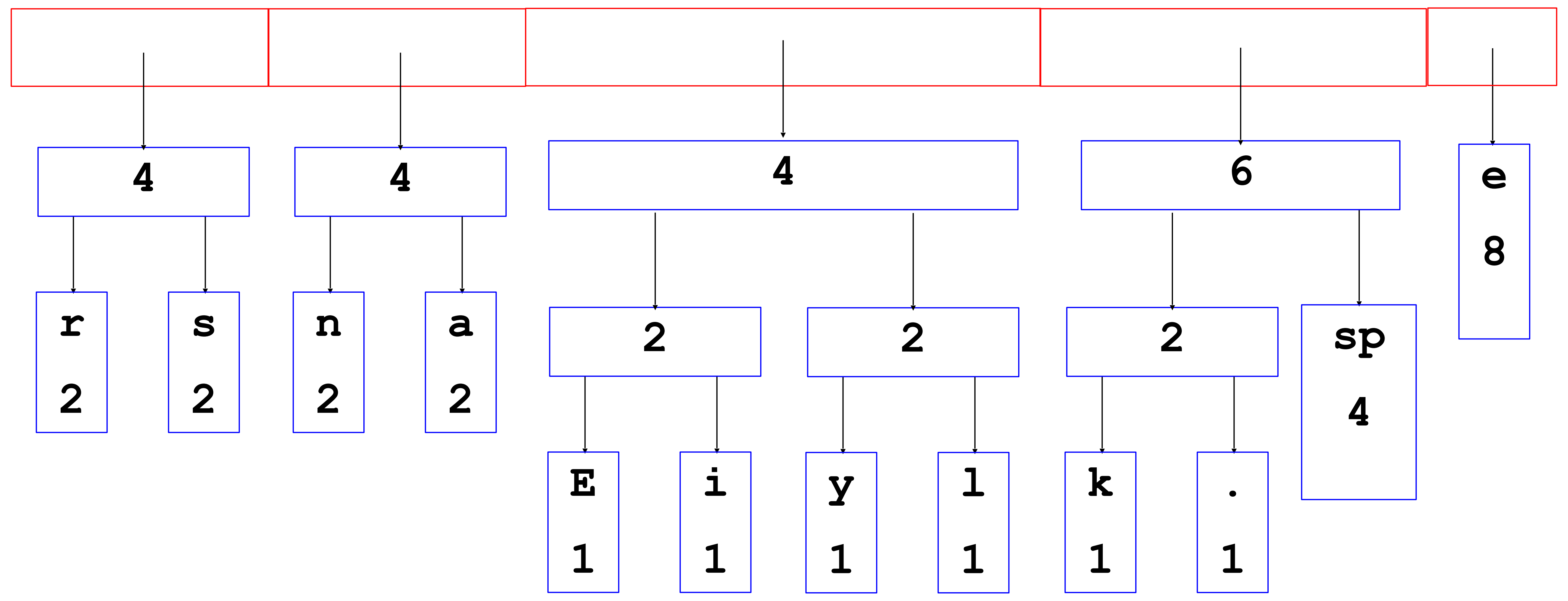
Building a tree



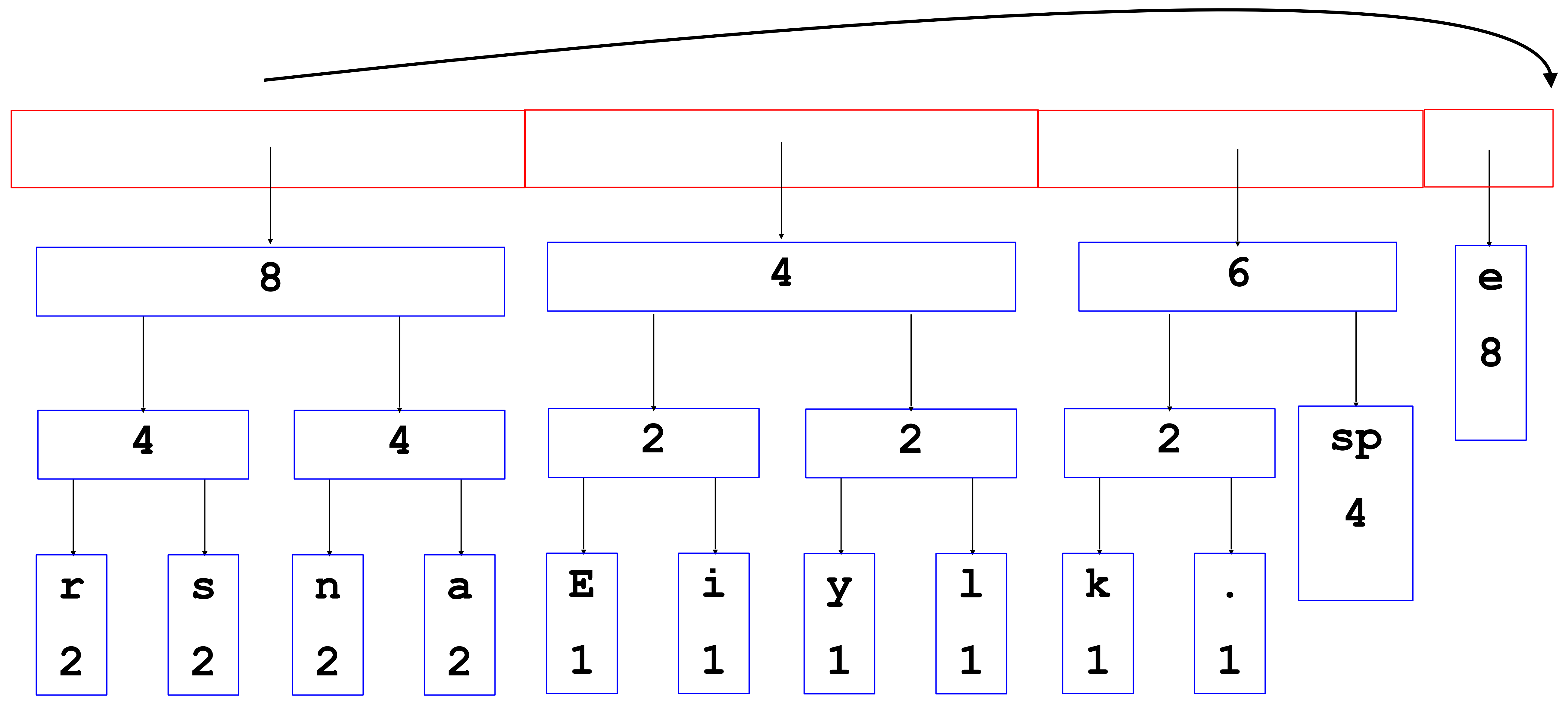
Building a tree



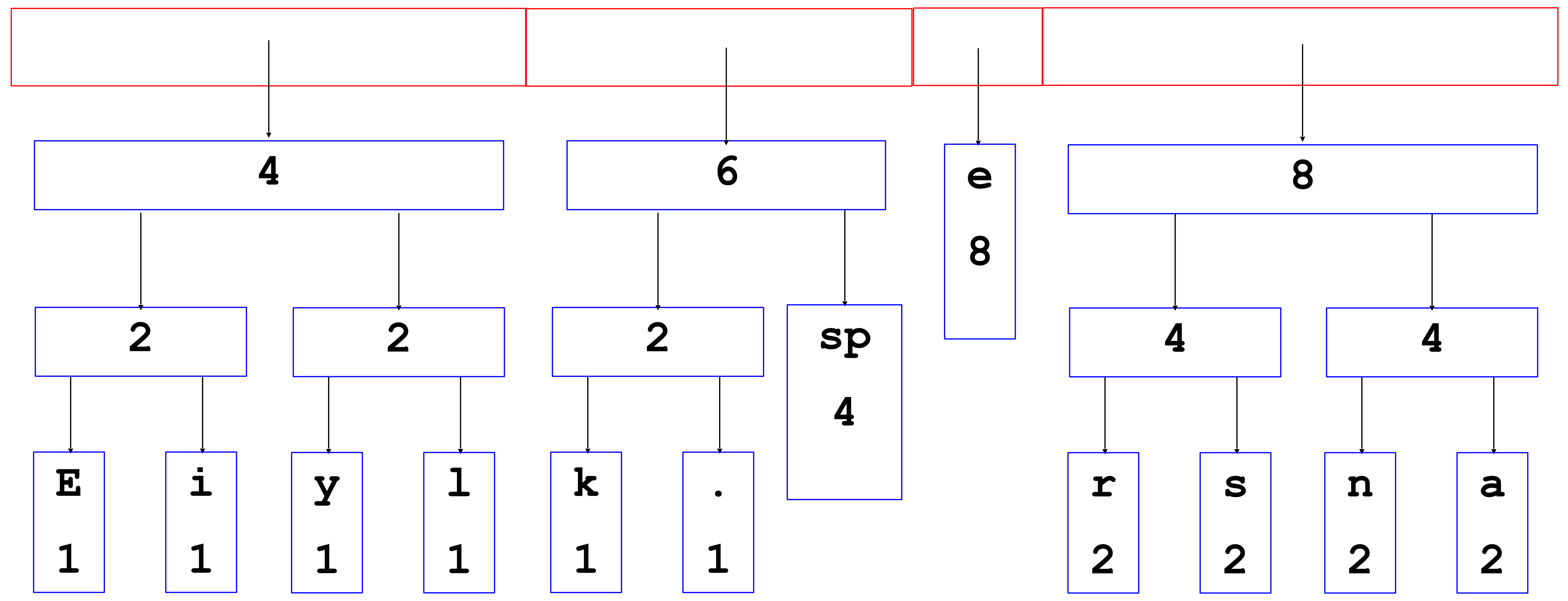
Building a tree



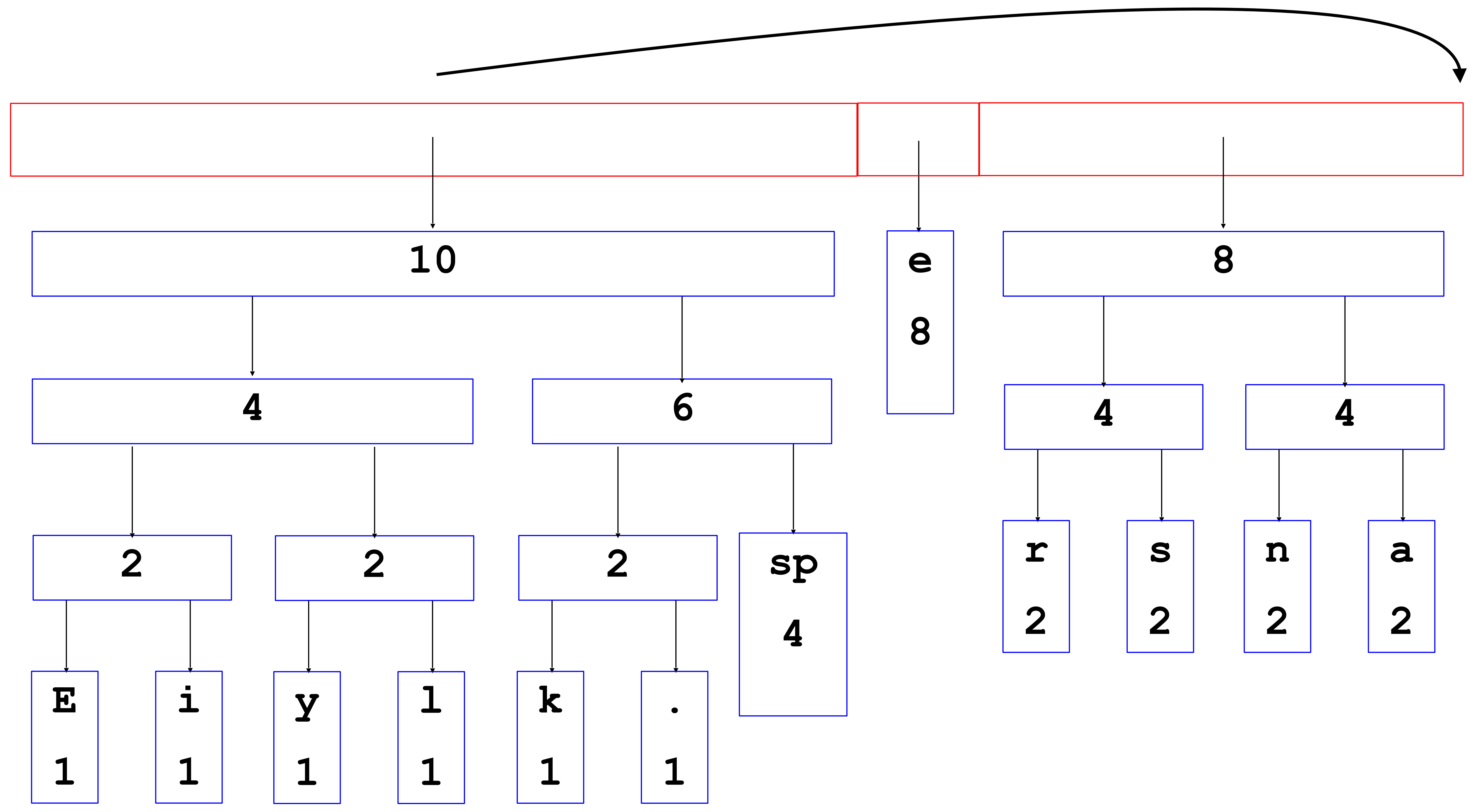
Building a tree



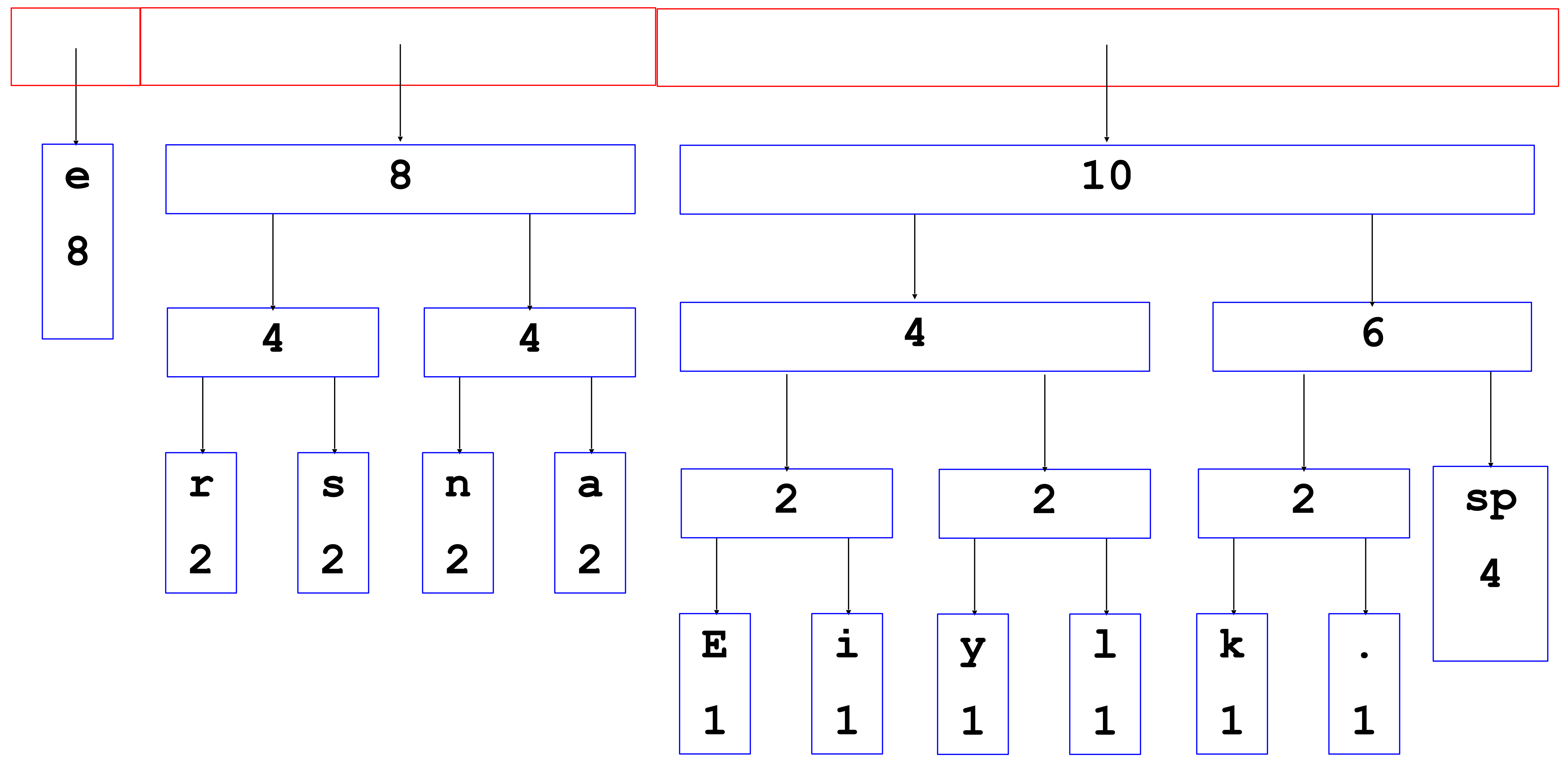
Building a tree



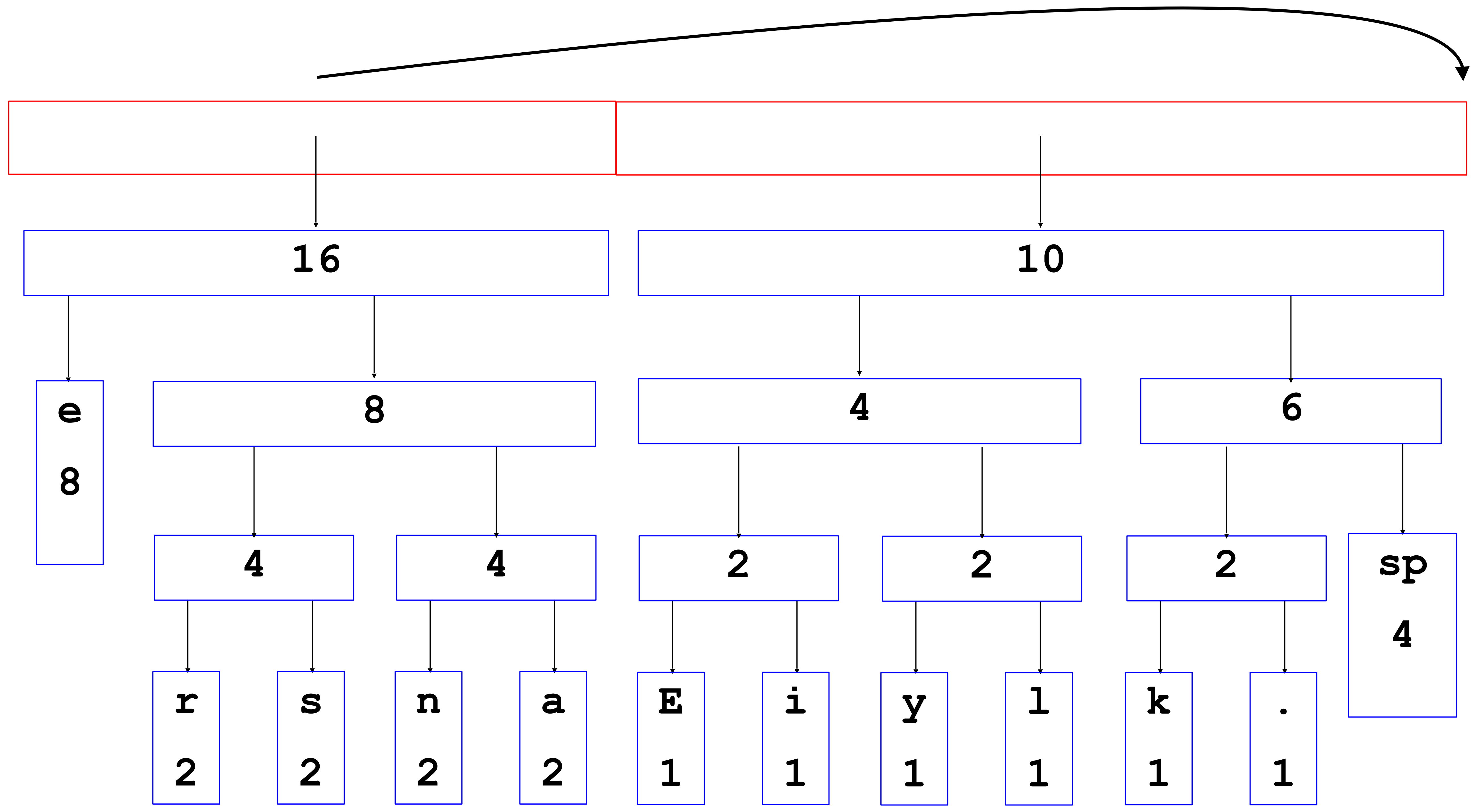
Building a tree



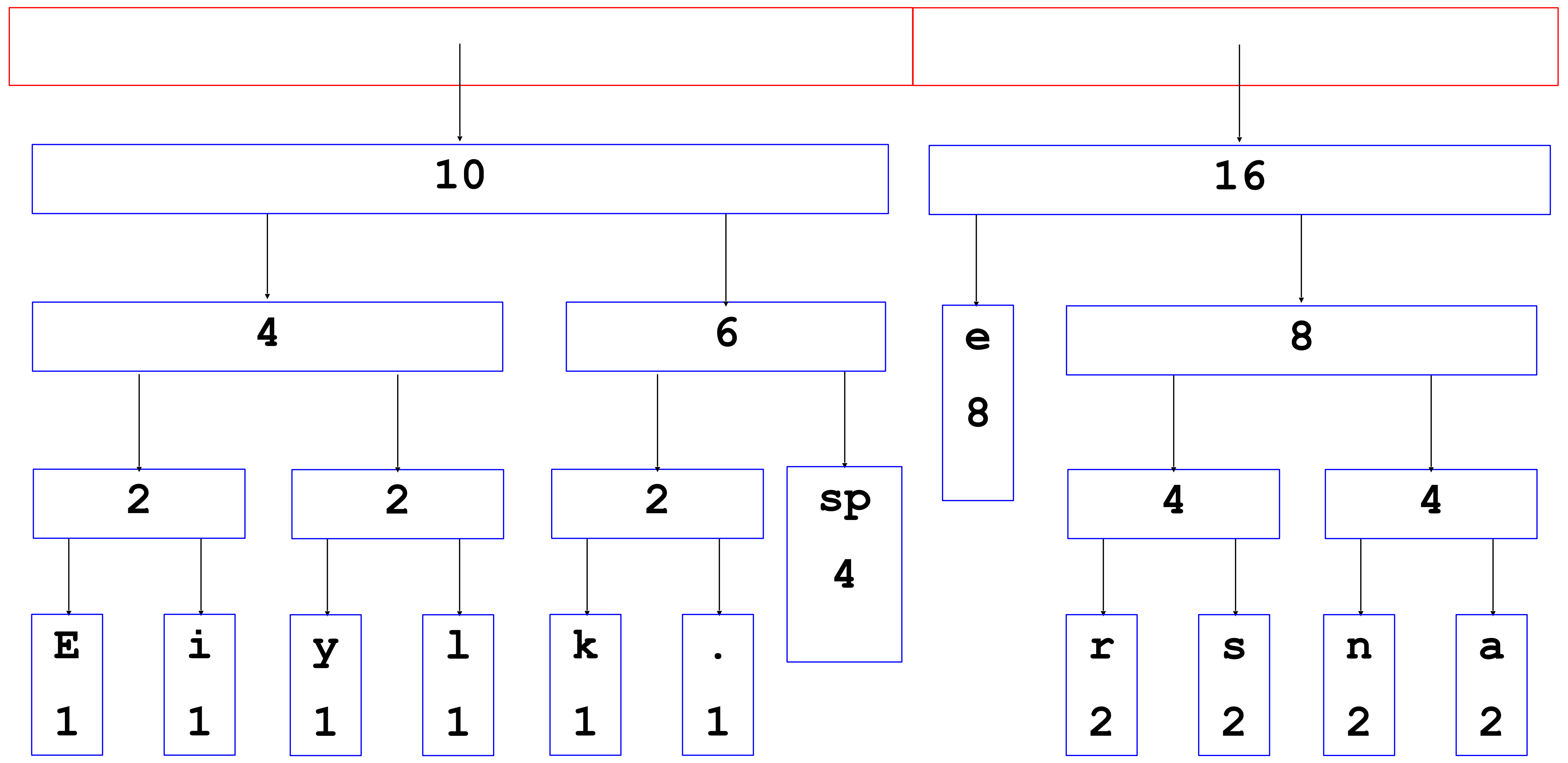
Building a tree



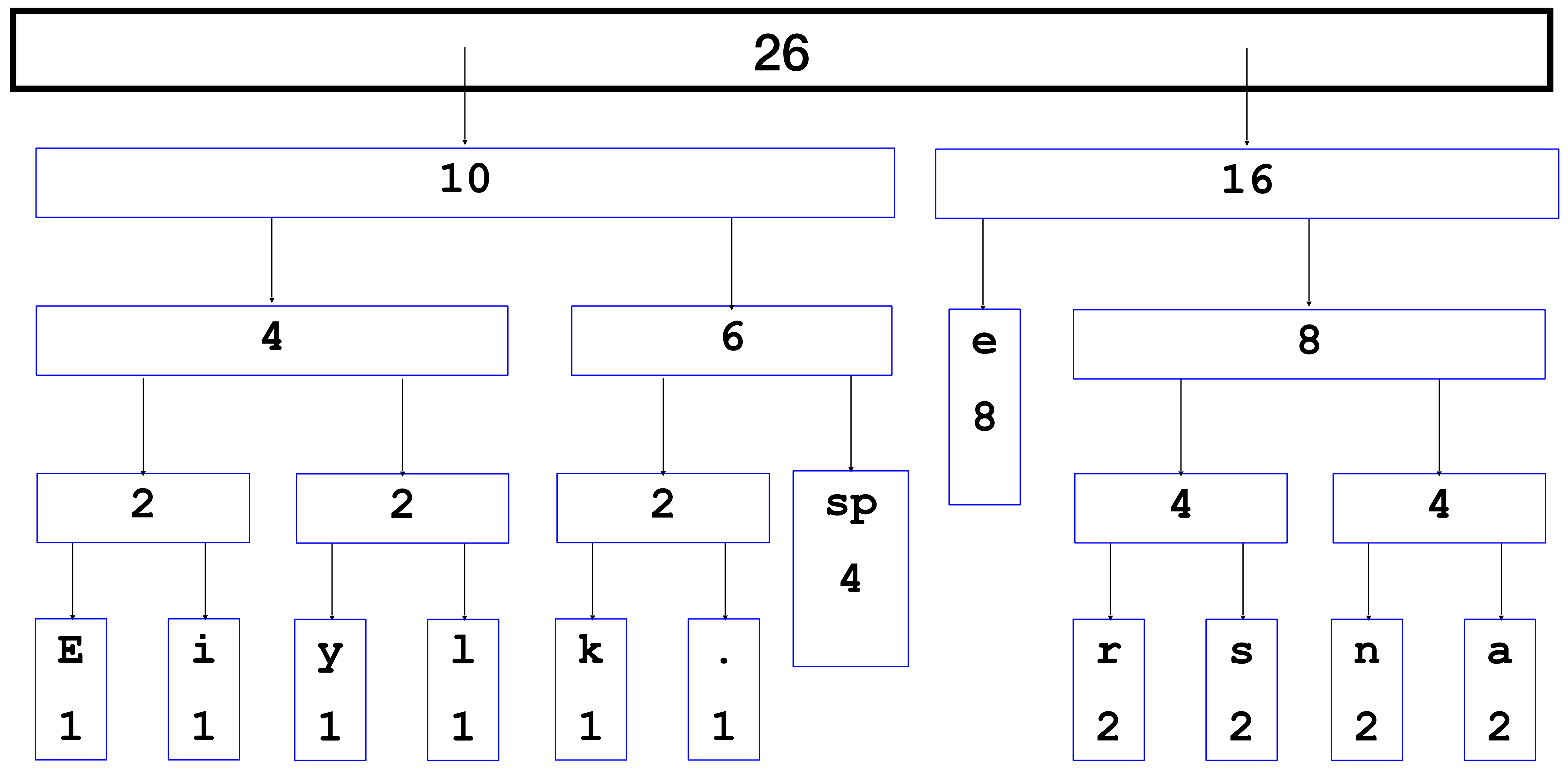
Building a tree



Building a tree

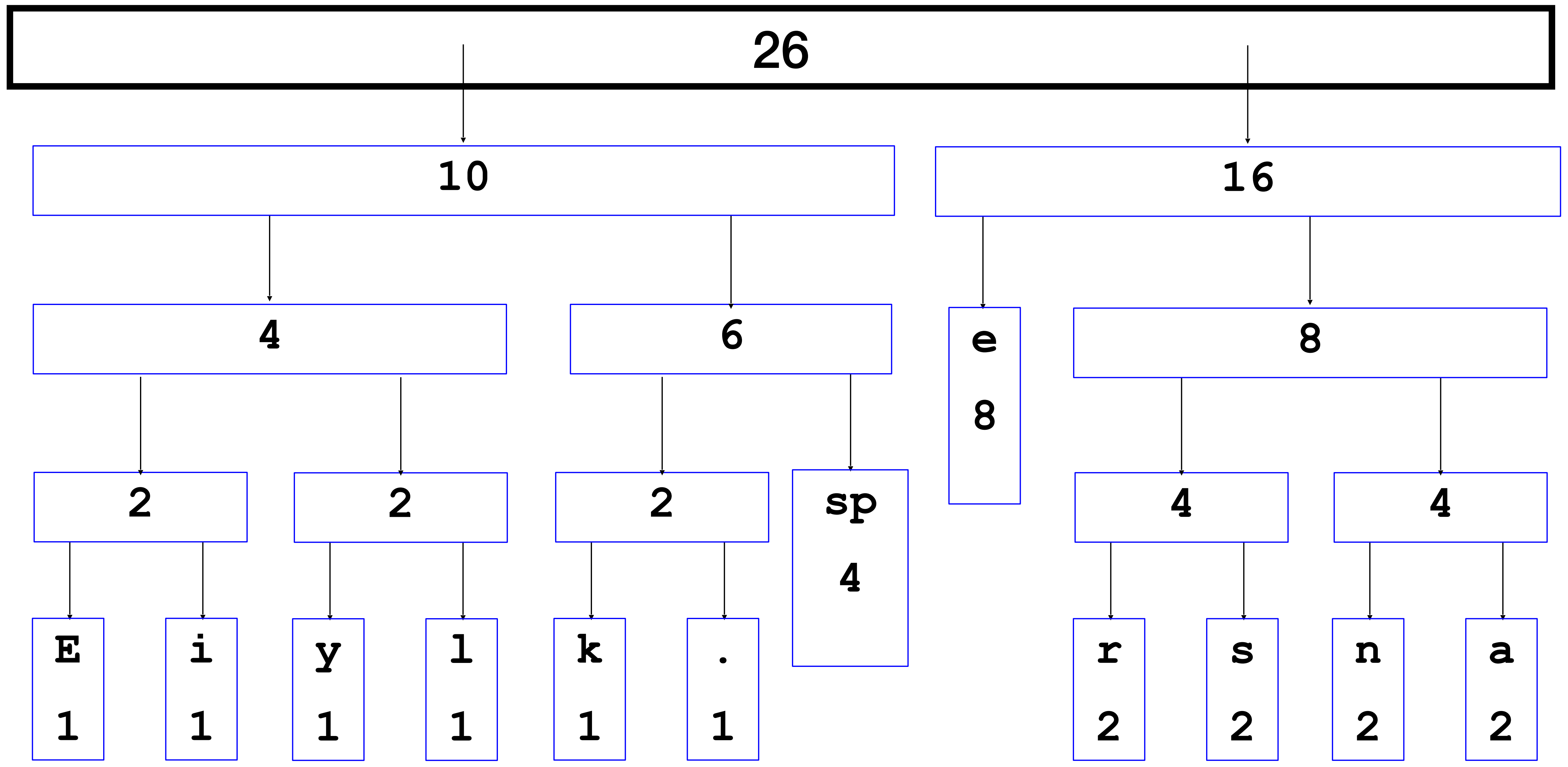


Building a tree



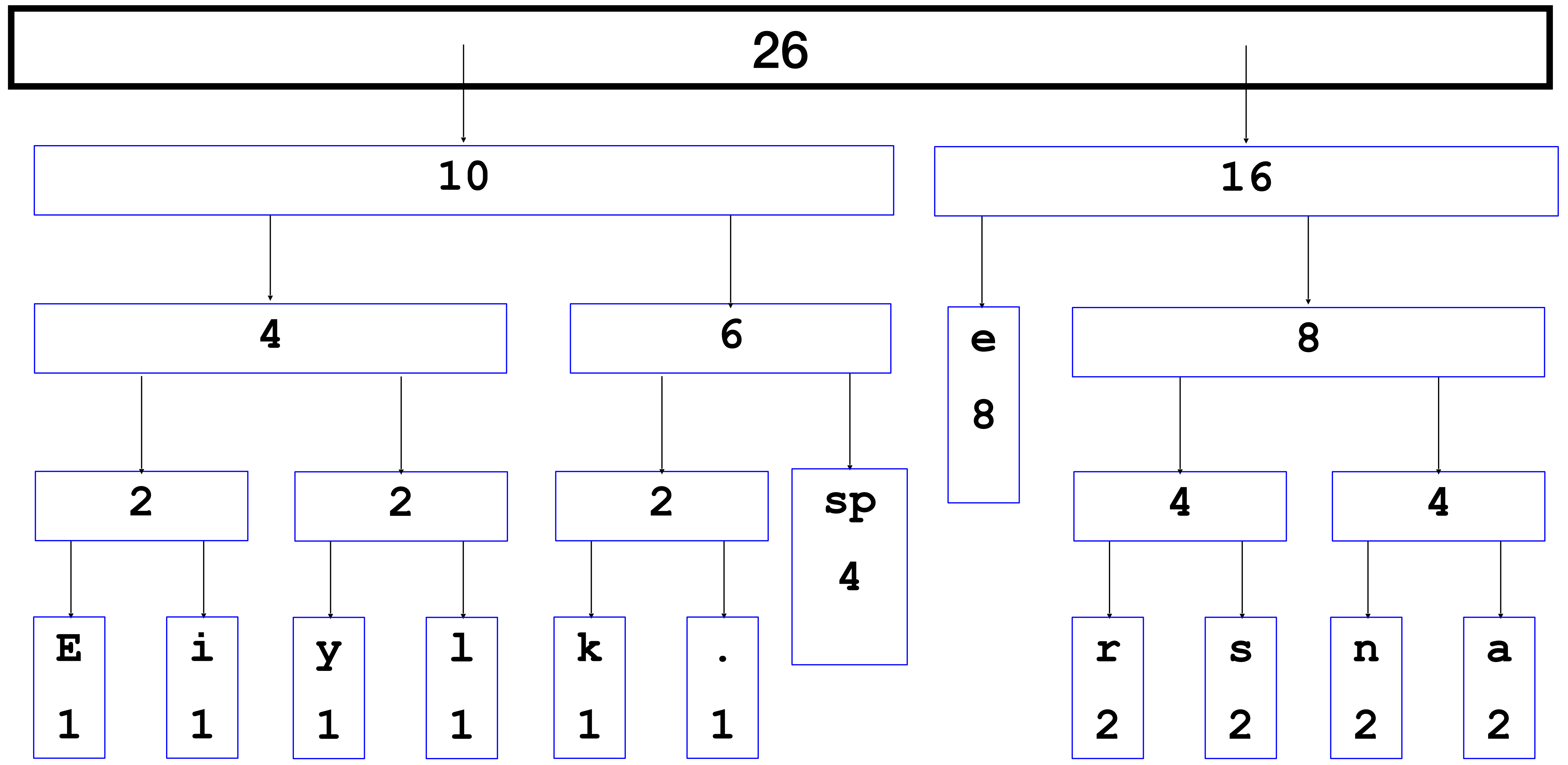
Algorithm

1. Scan text to be compressed and count occurrence of all characters
2. Sort or prioritize characters based on number of occurrences in text
3. Build Huffman code tree based on prioritized list
4. Perform a traversal of tree to determine all code words
5. Scan text again and create new file using the Huffman codes



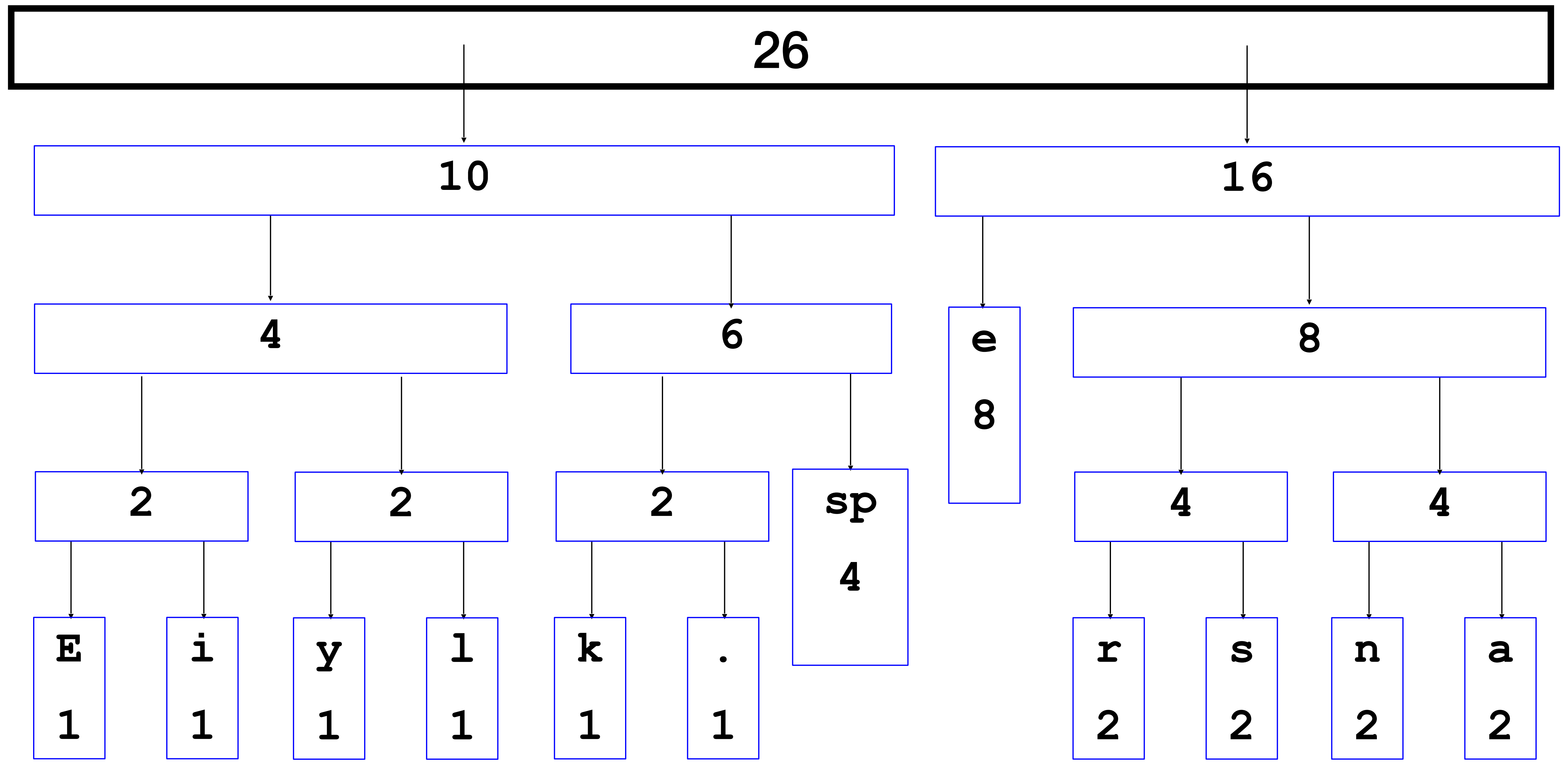
Traverse Tree for Codes

- Perform a traversal of the tree to obtain new code words
- Going left is a 0 going right is a 1
- code word is only completed when a leaf node is reached



Traverse Tree for Codes

Character	Code
E	0000
i	0001
y	0010
l	0011
k	0100
.	0101
“space”	011
e	10
r	1100
s	1101
n	1110
a	1111



Algorithm

1. Scan text to be compressed and count occurrence of all characters
2. Sort or prioritize characters based on number of occurrences in text
3. Build Huffman code tree based on prioritized list
4. Perform a traversal of tree to determine all code words
5. Scan text again and create new file using the Huffman codes

Character	Code
E	0000
i	0001
y	0010
l	0011
k	0100
.	0101
“space”	011
e	10
r	1100
s	1101
n	1110
a	1111

Eerie eyes seen near lake.

```
0000101100000110011100010101
1011010011111010111111000110
0111111010010010
```

Did we use improve the encoding?

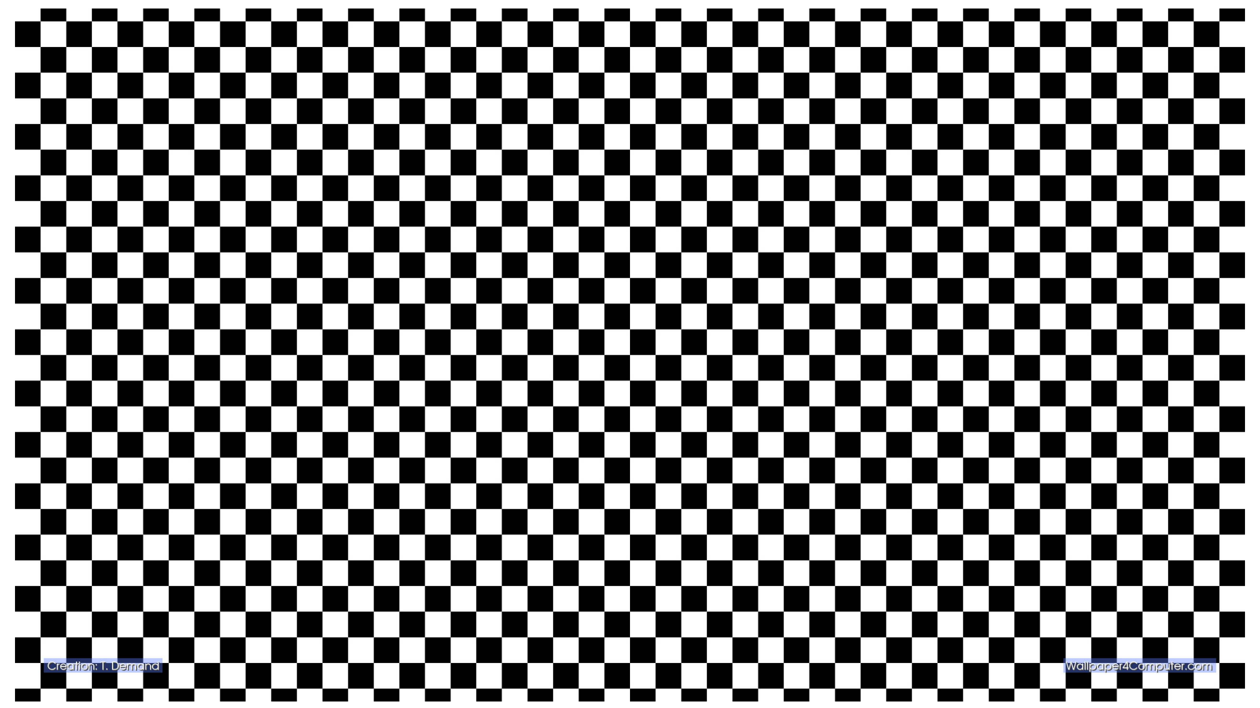
- ASCII would take 8 * 26 = 208 bits
- Naive encoding
  - 12 characters to encode
  - 4 bits are enough for each character
  - 4 * 26 = 104 bits
- Huffman encoding
  - 73 bits
  - average bits per character: 73/26≈ 2.81 $\boxed{\text{+ Also need to store the dictionary!}}$

Eerie eyes seen near lake.

```
0000101100
0001100111
0001010110
1101001111
1010111111
0001100111
1110100100
101
```

Summary

- Huffman coding is a technique used to compress files for transmission
- Uses statistical coding
- More frequently used symbols have shorter code words
- Works well for text and fax transmissions



- Does not work well for repeated pattern (for example alternate pixels picture)
- Further reading: LZ77 and LZ78 (Abraham **L**empel and Jacob **Z**iv) algorithms

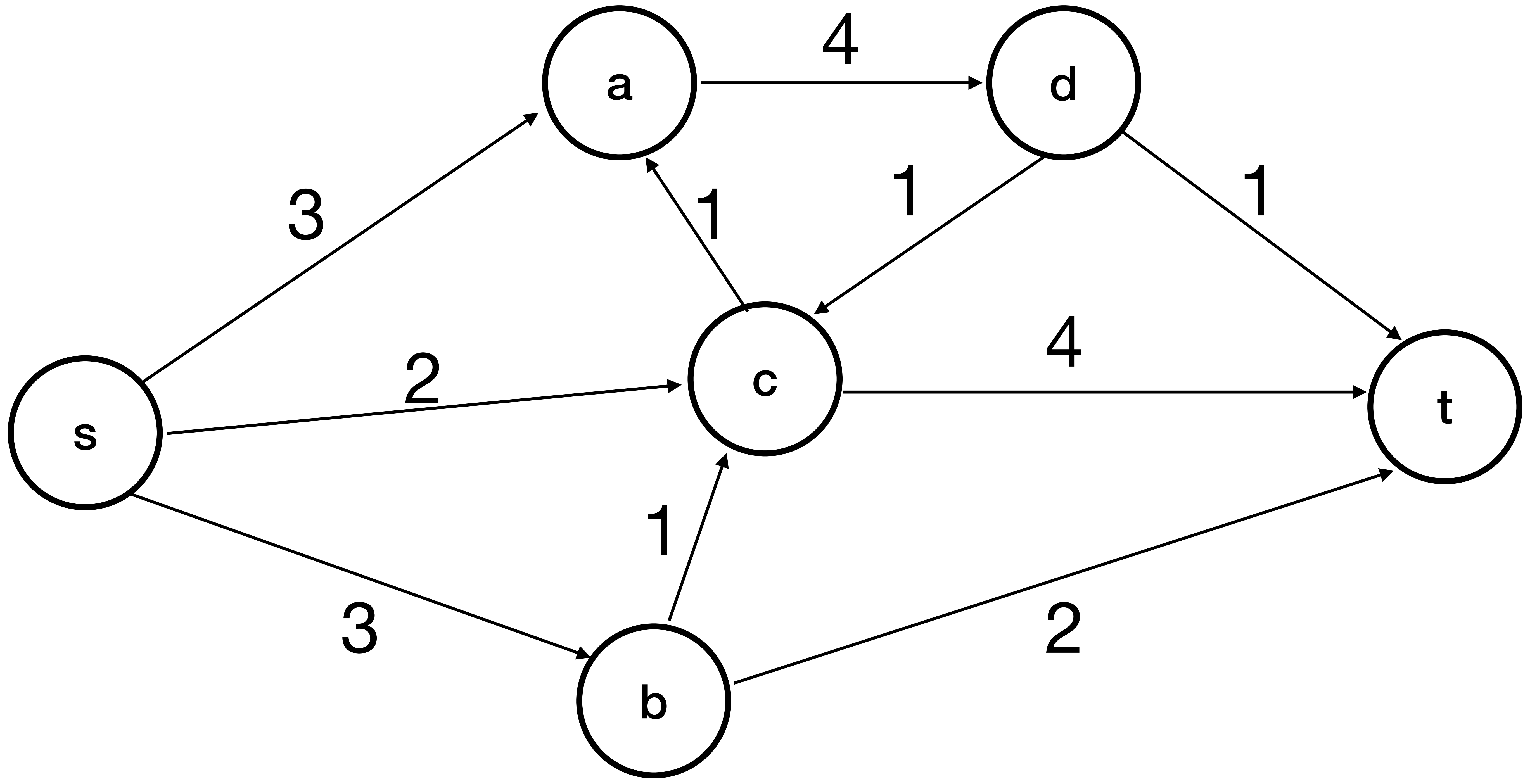
$\hookrightarrow$ Next week

Exercises

Max Flow Problem

1. Give the order of the augmenting paths you are considering

2. Apply Edmonds-Karp algorithm and give the graph (current flow) at each step



Huffman (encoding)

1. Encode the following text:

$$\boxed{\text{An illusory vision is a visionary illusion. Is it?}}$$

2. Give the corresponding encoding table
3. What is the average number of bits per character?

Huffman (decoding)

- Decode the following:

11000111101011010111000001111101

Character	Code
!	101
A	11
B	00
C	010
D	100
R	011