Data Structures in Java

Lecture 20: Algorithm Design Techniques

12/2/2015

Daniel Bauer

Algorithms and Problem Solving

- Purpose of algorithms: find solutions to problems.
- Data Structures provide ways of organizing data such that problems can be solved more efficiently
  - Examples: Hashmaps provide constant time access by key, Heaps provide a cheap way to explore different possibilities in order…
- When confronted with a new problem, how do we:
  - Get an idea of how difficult it is?
  - **Develop an algorithm to solve it?**

Common Types of Algorithms

- Greedy Algorithms
- Divide and Conquer
- Dynamic Programming

- We have already seen some examples for each.
- We will look at the general techniques and some additional examples.

Greedy Algorithms

"Take what you can get now"

- Algorithm uses multiple "phases" or "steps". In each phase a local decision is made that appears to be good.
- Making a local decision is fast (often O(log N) time). Examples: Dijkstra's, Prim's, Kruskal's
- Greedy algorithms assume that making ***locally optimal decisions*** leads to a ***global optimum.***
  - This works for some problems.
  - For many others it doesn't. Greedy algorithms are still useful to find approximate solutions.

ASCII Encoding

Character	Decimal	Binary
⋮		
A	65	1000001
B	66	1000010
C	67	1000011
D	68	1000100
E	69	1000101
⋮		
a	97	1100000
b	98	1100001
c	99	1100010
d	100	1100011
e	101	1100100
⋮		

- The ASCII codec contains 128 characters (about 100 printable characters + special chars).
- Each character needs $\lceil \log 128 \rceil = 7$ bits of space.

Can we store data more efficiently?

A 5-Character Alphabet

Character	Decimal	Binary
a	0	“000"
e	1	“001"
i	2	“010"
s	3	“011"
t	4	“100"
space	5	“101"
newline	6	“110"

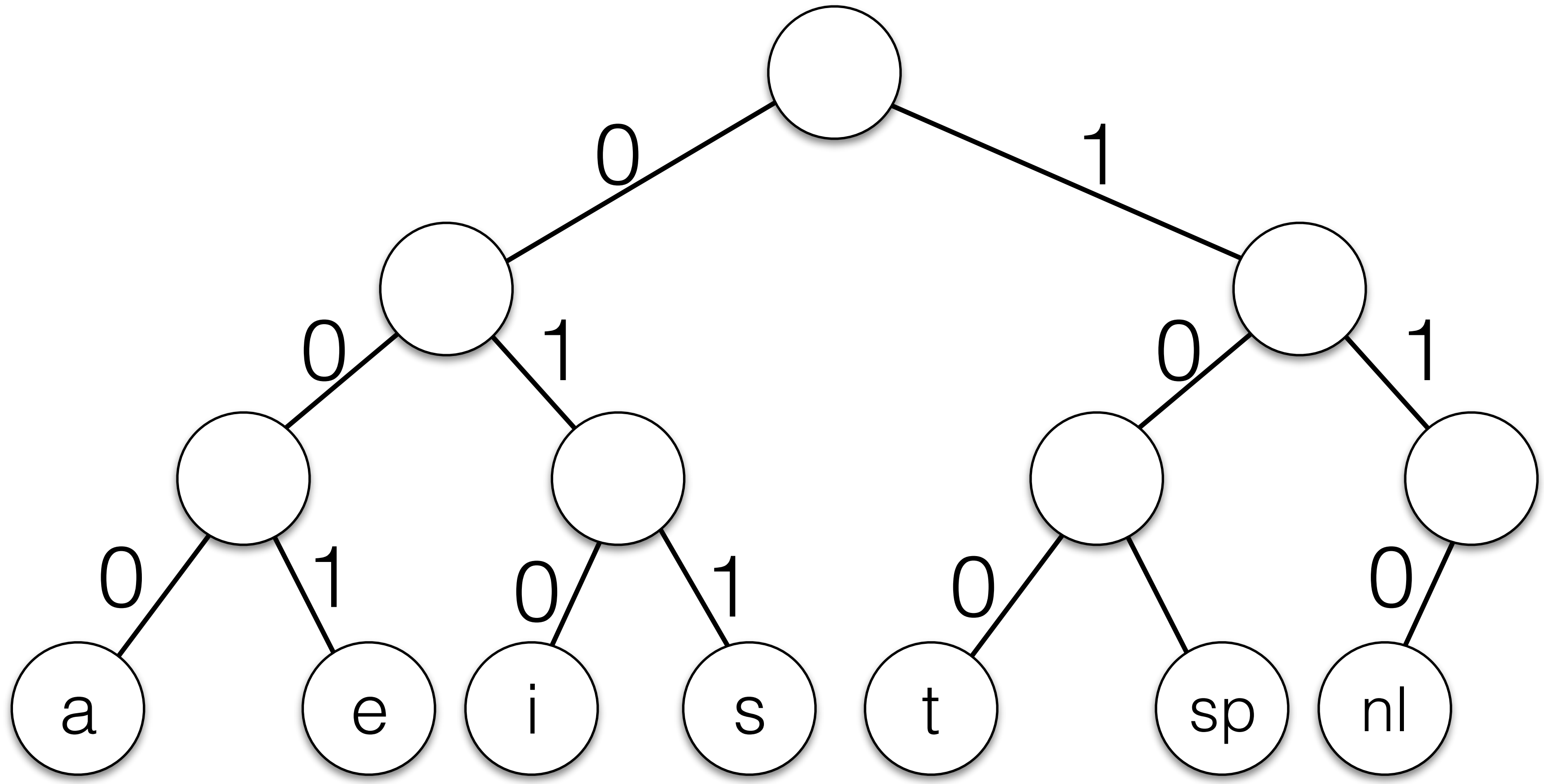
A 5-Character Alphabet

Assume we see each character with a certain frequency in a textfile. We can then compute the total number of bits required to store the file.

Character	Decimal	Binary Code	Frequency	Total bits
a	0	"000"	10	30
e	1	"001"	15	45
i	2	"010"	12	36
s	3	"011"	3	9
t	4	"100"	4	12
space	5	"101"	13	39
newline	6	"110"	1	3
				Total: 175

Prefix Trees

Character	Binary
a	“000"
e	“001"
i	“010"
s	“011"
t	“100"
space	“101"
newline	“110"



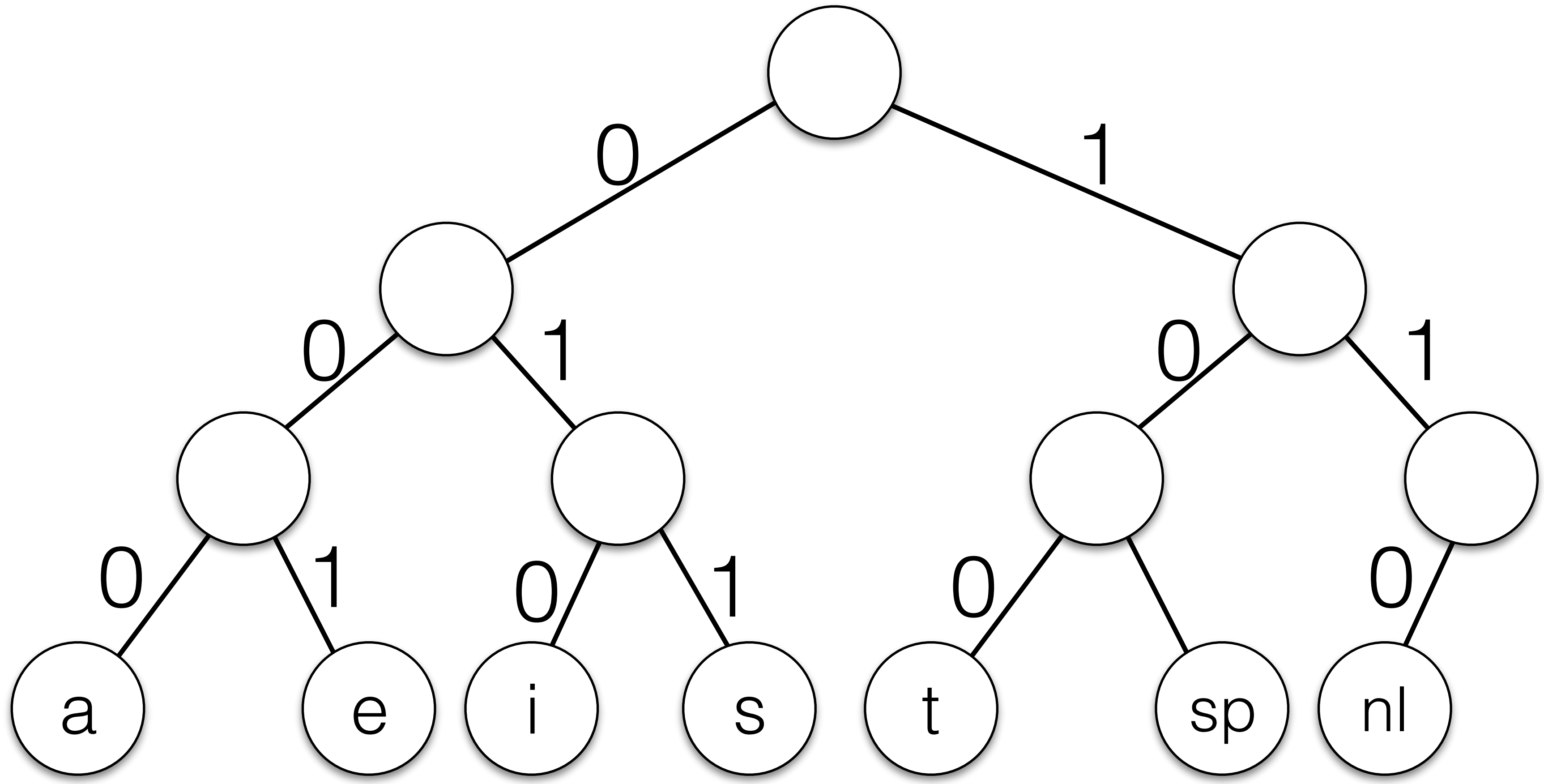
d_i depth of character i

f_i frequency of i in the file

$$\text{file size} = \sum_i d_i f_i$$

Prefix Trees

Character	Binary
a	“000"
e	“001"
i	“010"
s	“011"
t	“100"
space	“101"
newline	“110"



d_i depth of character i

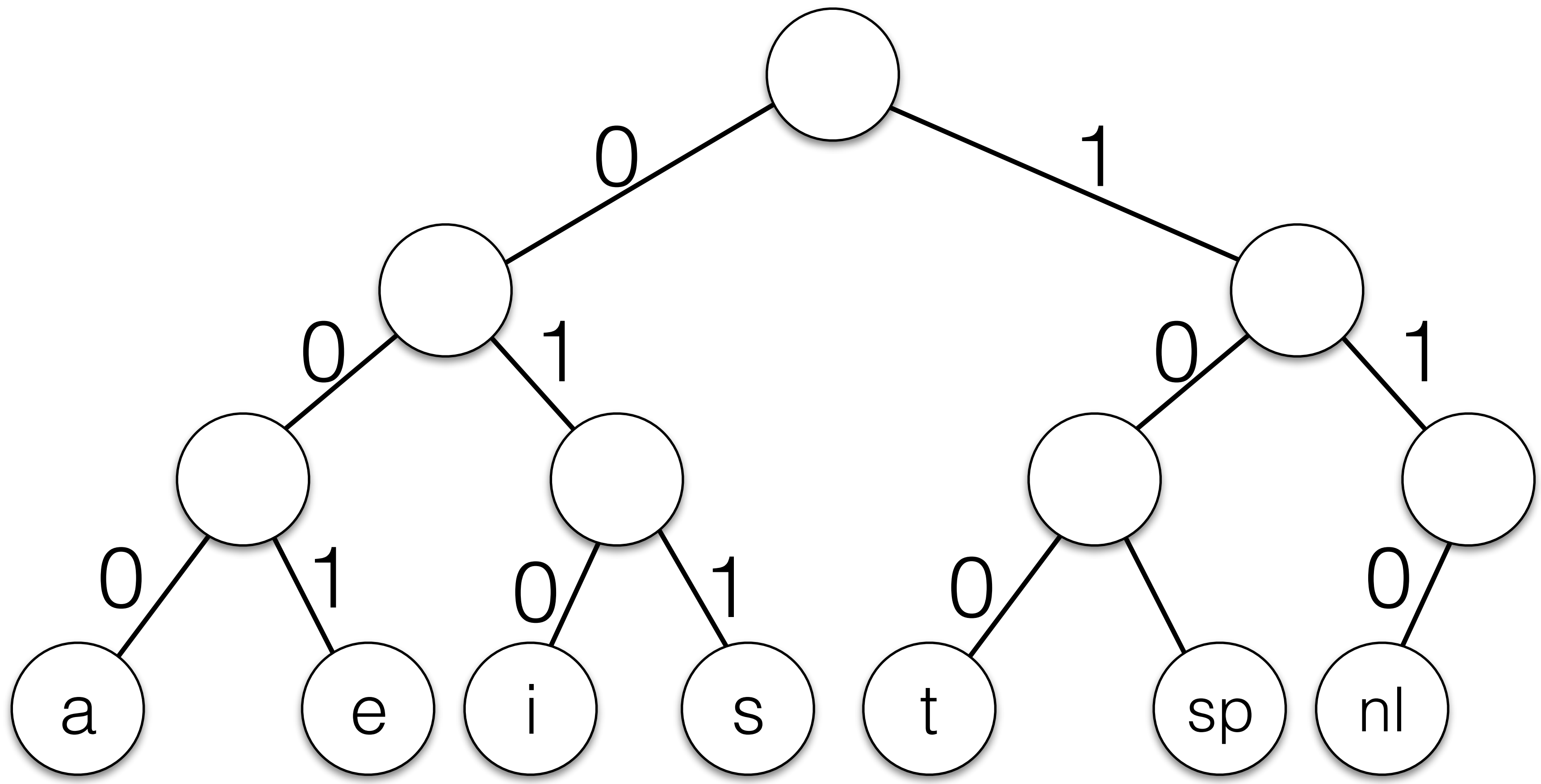
$$\text{file size} = \sum_i d_i f_i$$

f_i frequency of i in the file

Can we restructure the tree to minimize the file size?

Prefix Trees

Character	Binary
a	“000"
e	“001"
i	“010"
s	“011"
t	“100"
space	“101"
newline	“110"



d_i depth of character i

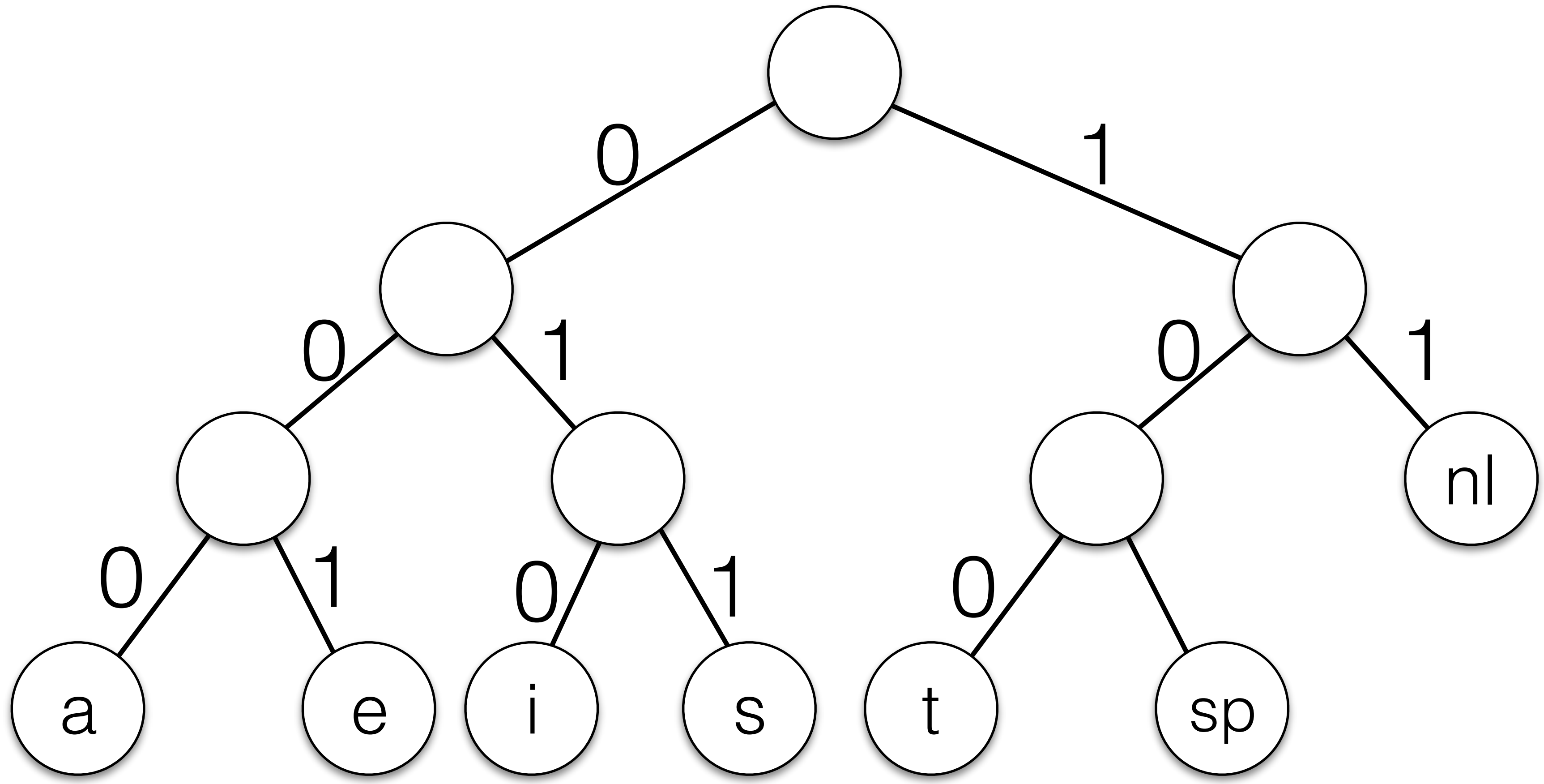
$$\text{file size} = \sum_i d_i f_i$$

f_i frequency of i in the file

Prefix “11” is not used for any other character than *nl.*

Prefix Trees

Character	Binary
a	“000"
e	“001"
i	“010"
s	“011"
t	“100"
space	“101"
newline	“11"



d_i depth of character i

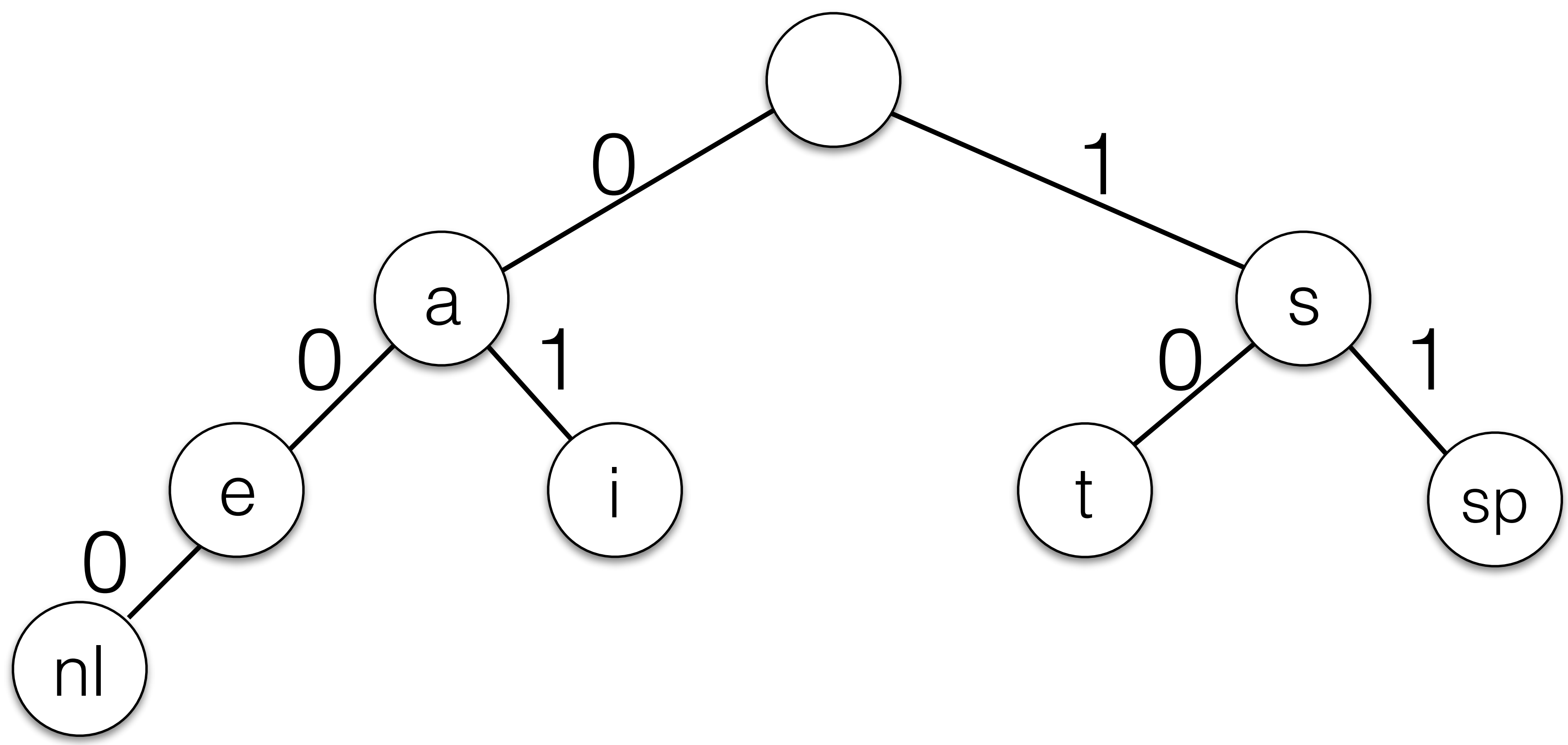
$$\text{file size} = \sum_i d_i f_i$$

f_i frequency of i in the file

Prefix “11” is not used for any other character than *nl.*

Prefix Trees

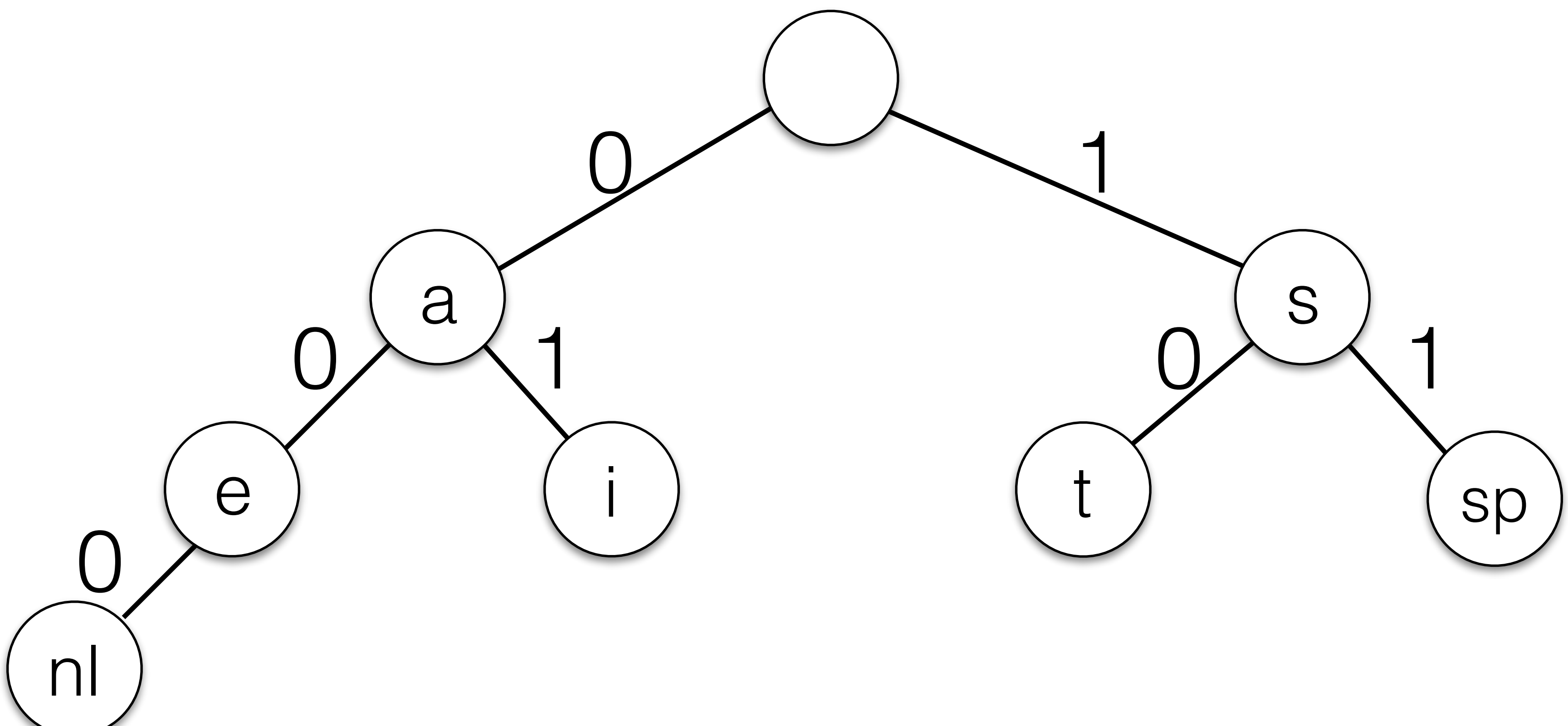
We cannot place characters on interior nodes, or else encoded sequences would be ambiguous.



000110

Prefix Trees

We cannot place characters on interior nodes, or else encoded sequences would be ambiguous.

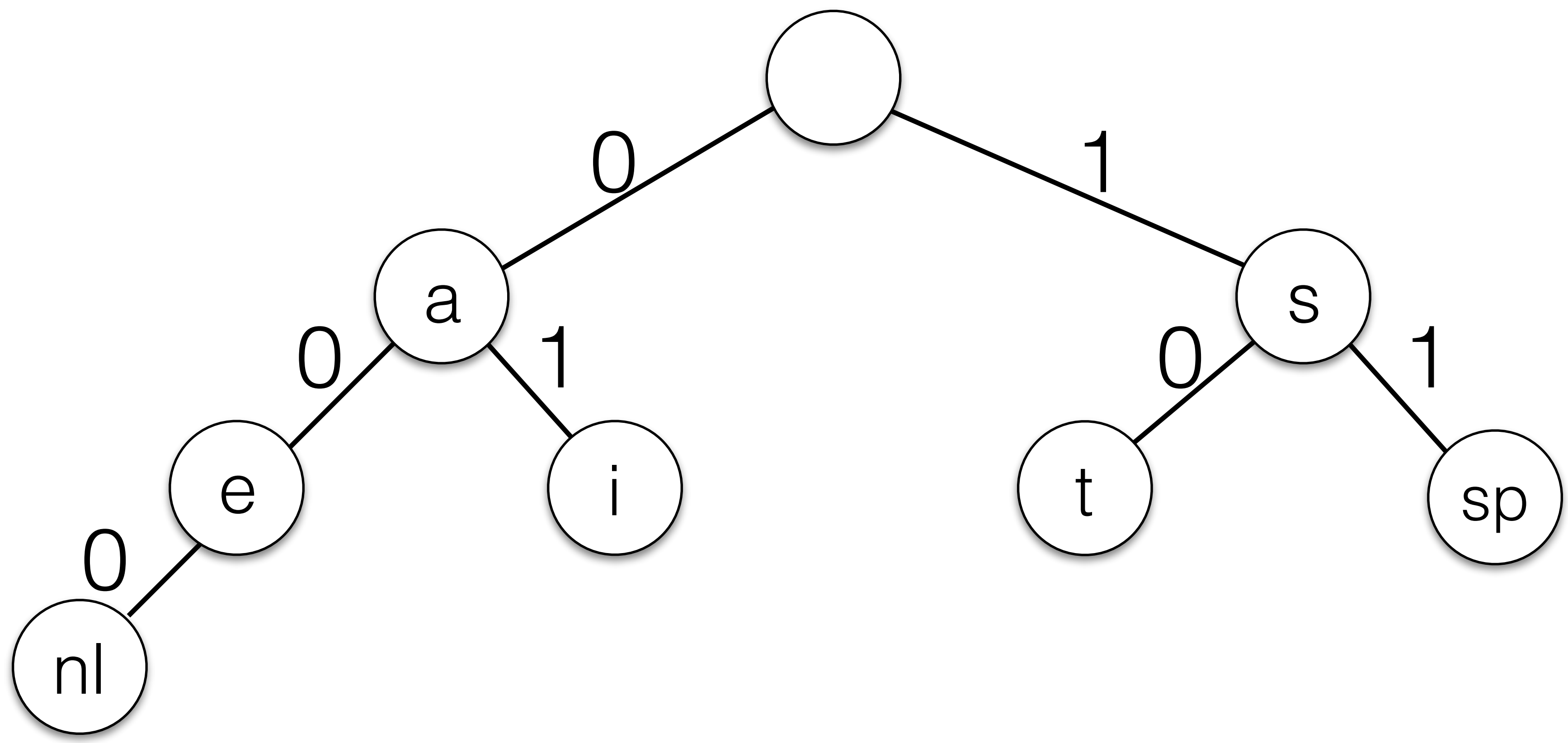


00_01_**10**

e i t

Prefix Trees

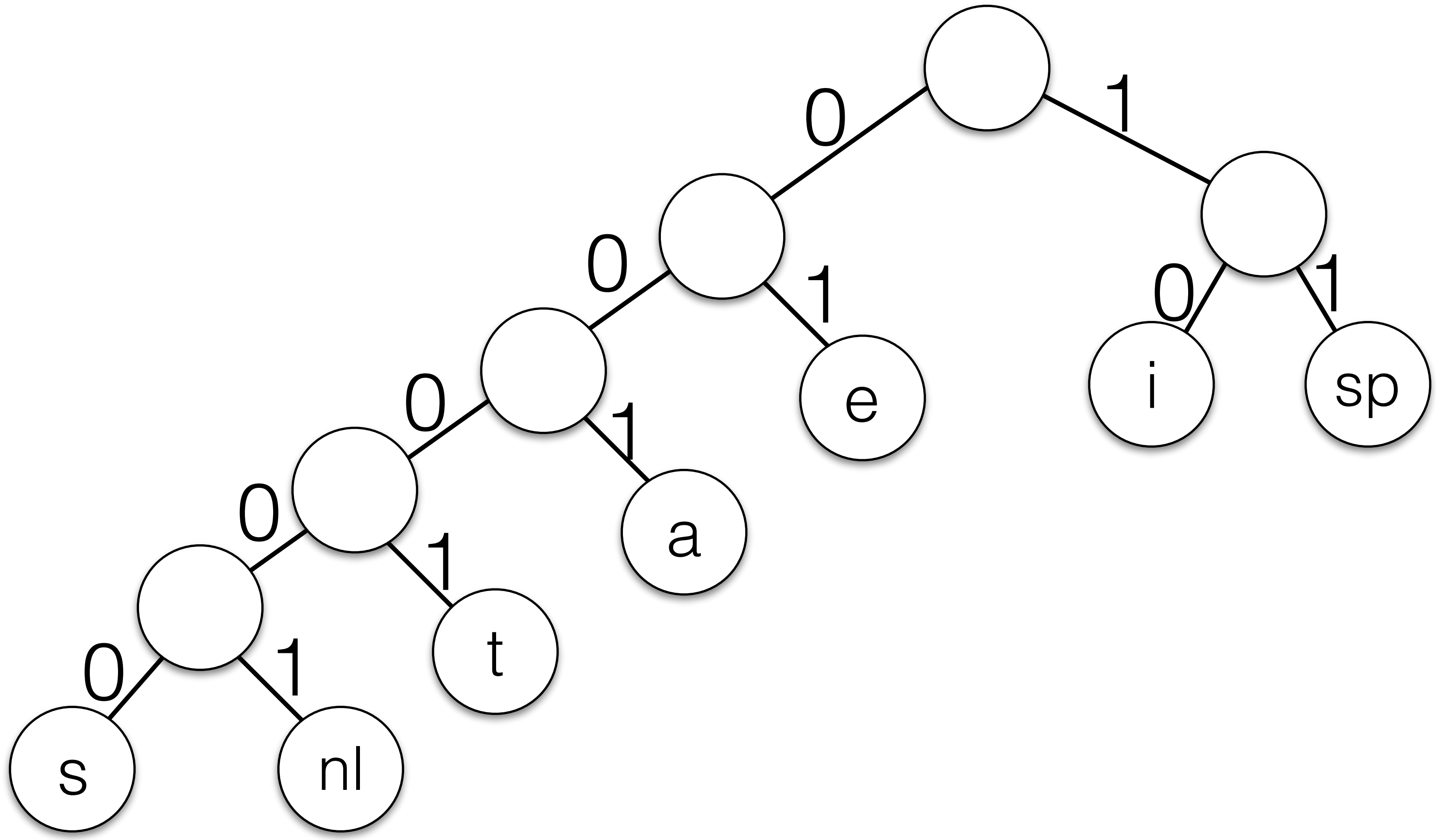
We cannot place characters on interior nodes, or else encoded sequences would be ambiguous.



000 *1 1* **0**
nl sp t

Huffman Code

chr	bin	f_i
e	“01"	15
sp	“11"	13
i	“10"	12
a	“001”	10
t	“0001”	4
s	“00000”	3
nl	“00001"	1



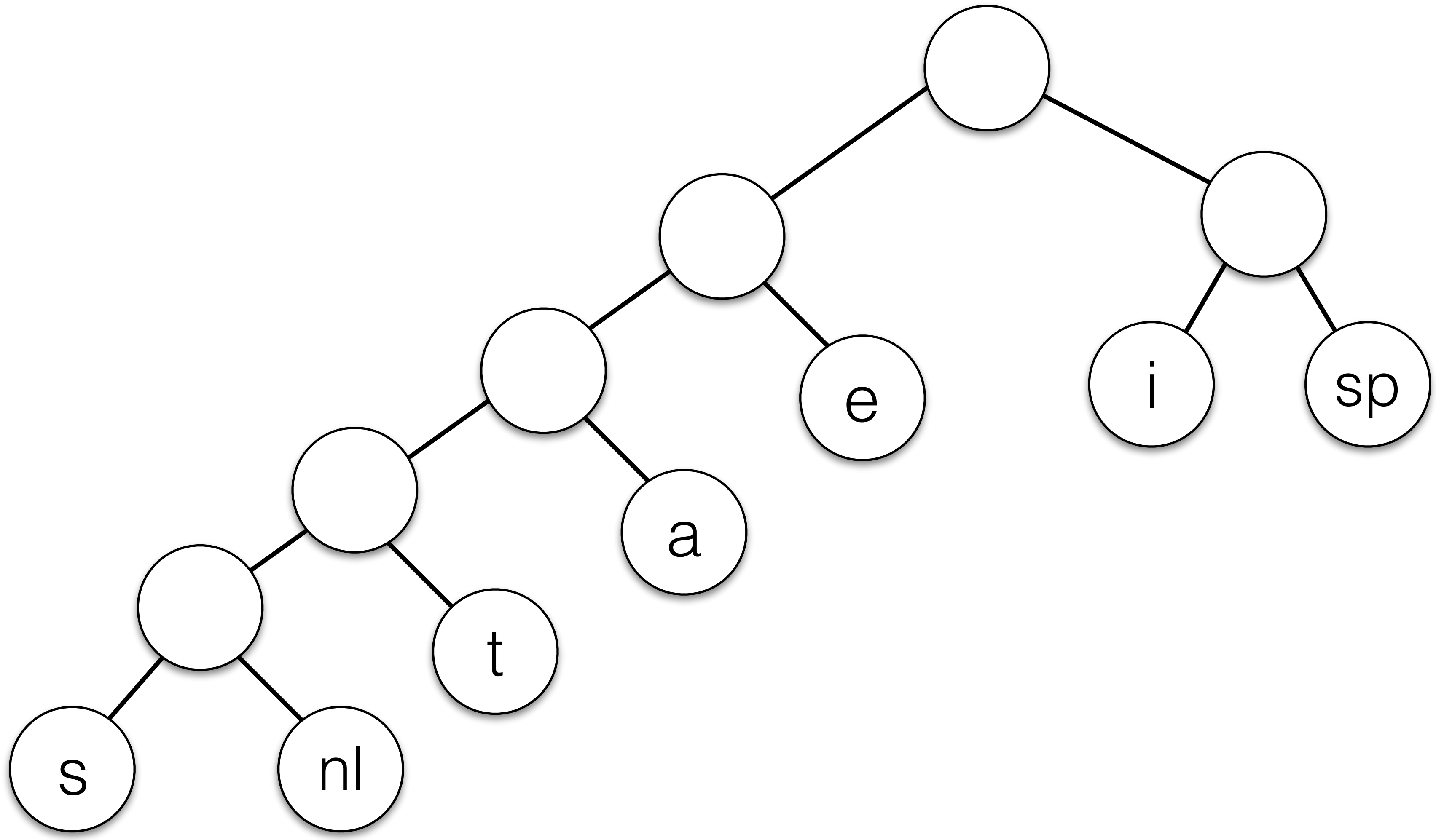
$$\text{file size} = \sum_i d_i f_i$$

- All characters are at leaves.
- Frequent characters have short codes.
- Rare characters have long codes.

Huffman Code

chr	bin	f_i
e	“01"	15
sp	“11"	13
i	“10"	12
a	“001”	10
t	“0001”	4
s	“00000”	3
nl	“00001"	1

Total size: 146



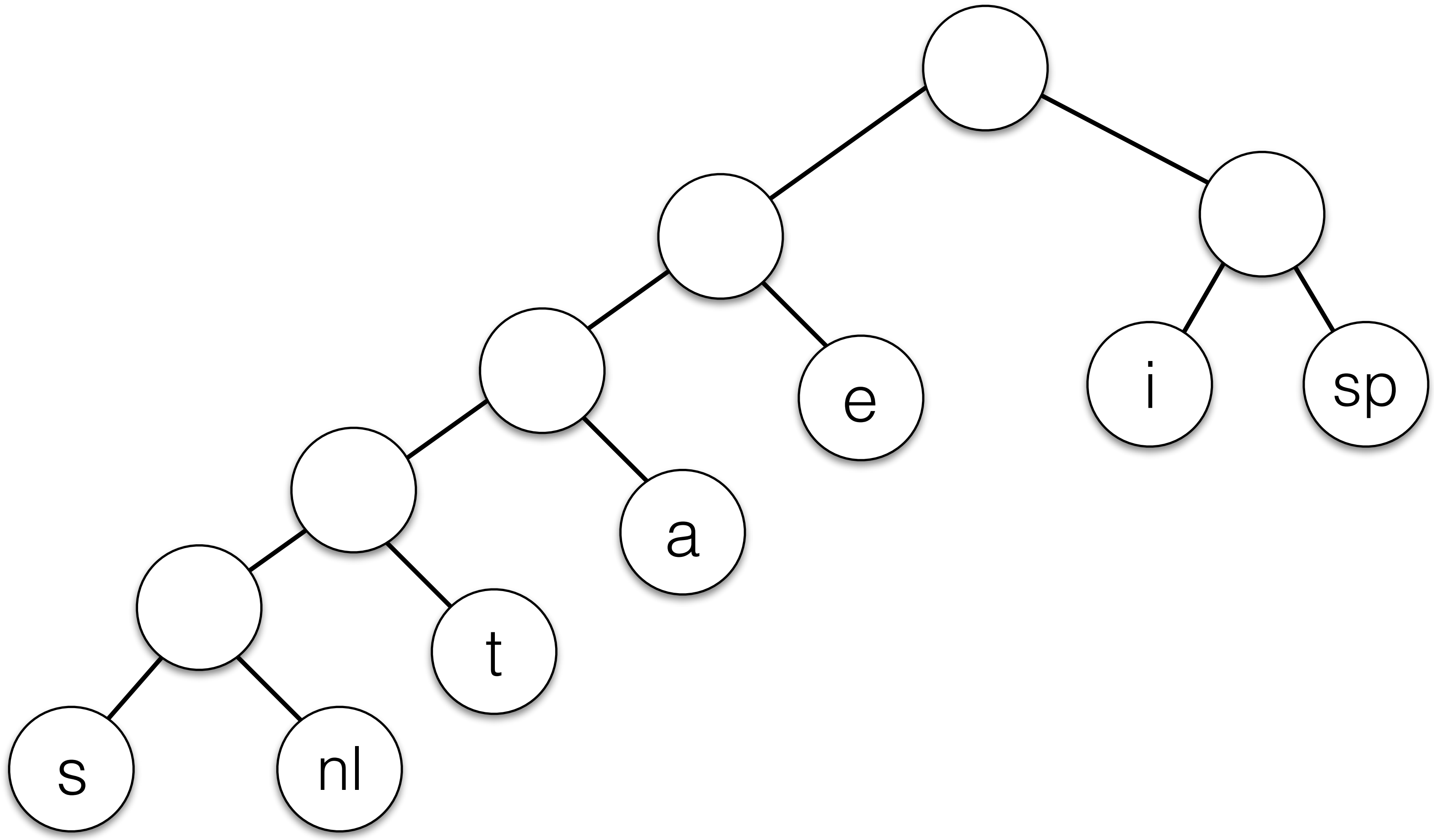
0000001001000100001

- This example: Save 16% space compared to standard coding.
- Typically much better compression (for larger files and alphabets).

Huffman Code

chr	**bin**	**f_i**
e	“01"	15
sp	“11"	13
i	“10"	12
a	“001”	10
t	“0001”	4
s	“00000”	3
nl	“00001"	1

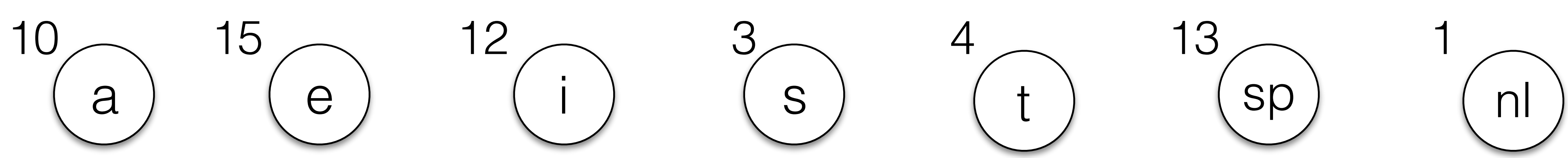
Total size: 146



0000001**001**0001**00001**

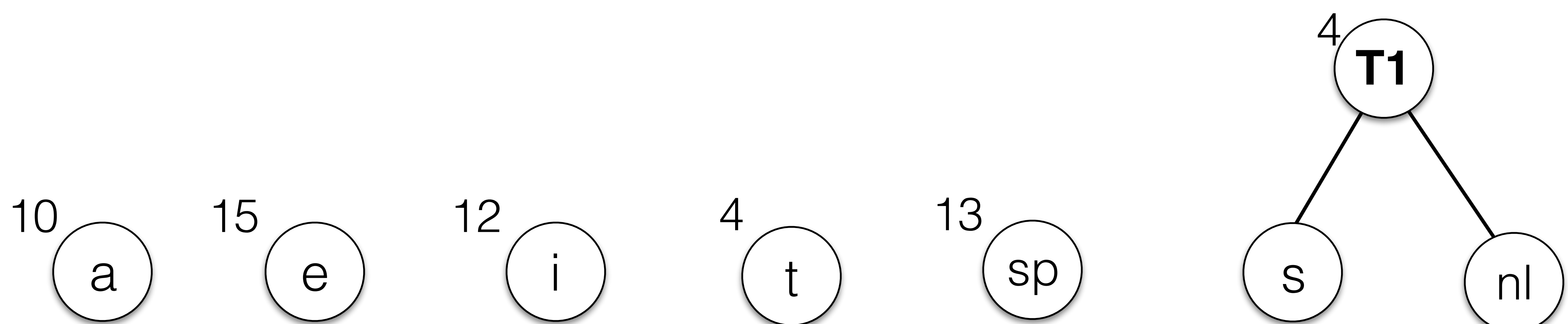
- This example: Save 16% space compared to standard coding.
- Typically much better compression (for larger files and alphabets).

Huffman's Algorithm



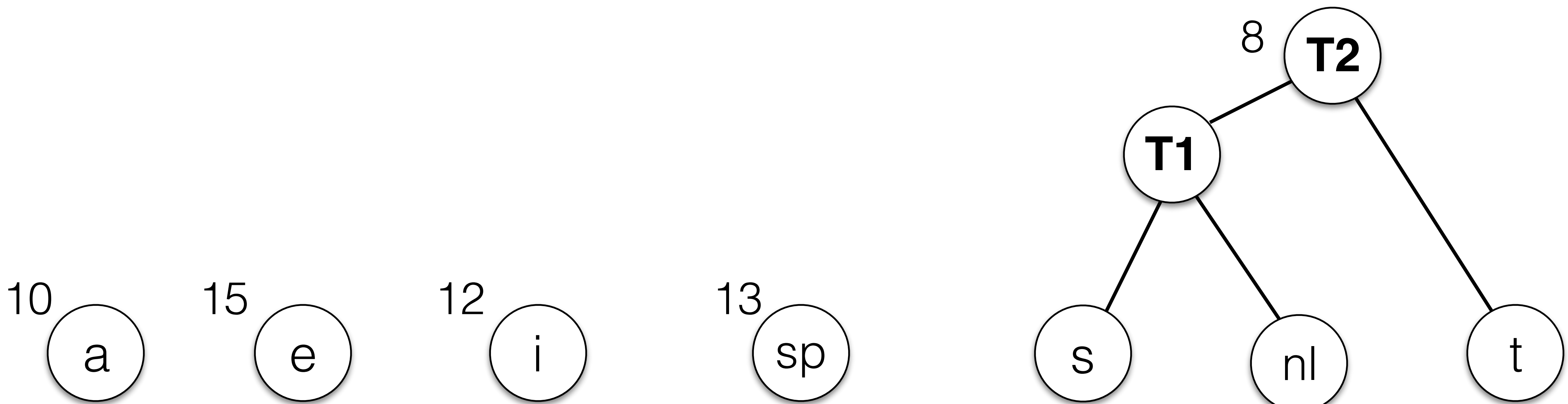
- Maintain a forest of prefix trees.
- Weight of a tree T = sum of frequencies of characters in T.

Huffman's Algorithm



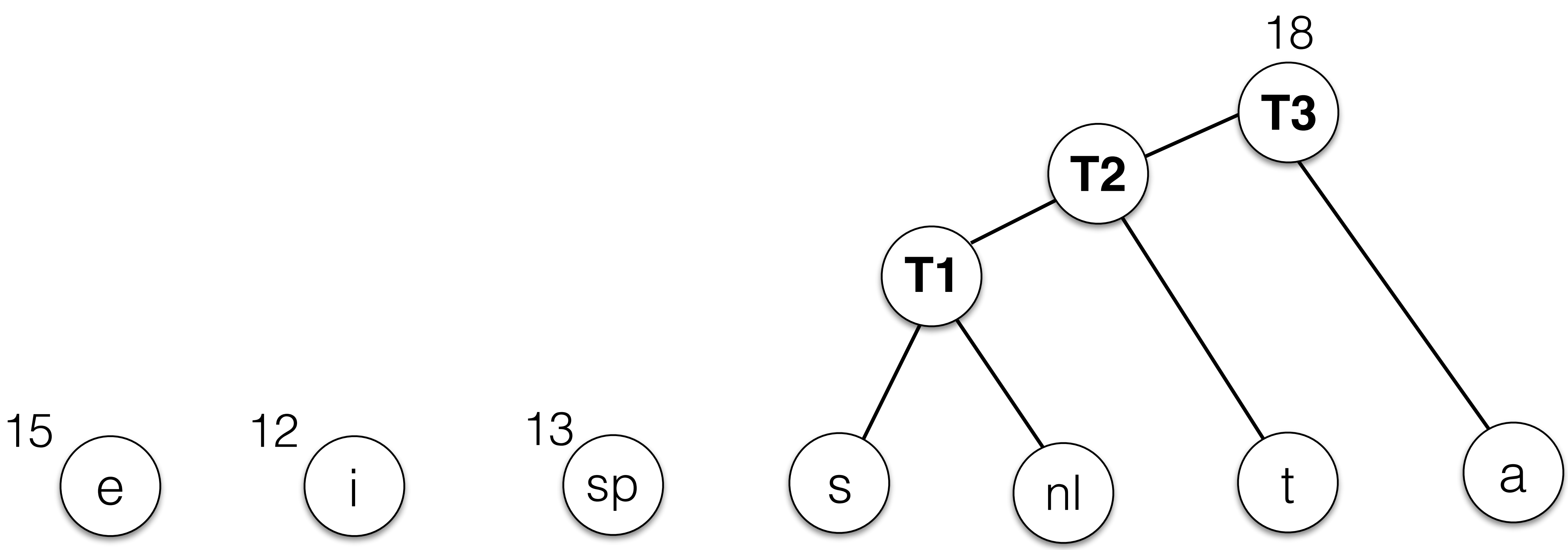
- In every phase:
  - Choose the two trees with smallest weight and merge them.

Huffman's Algorithm



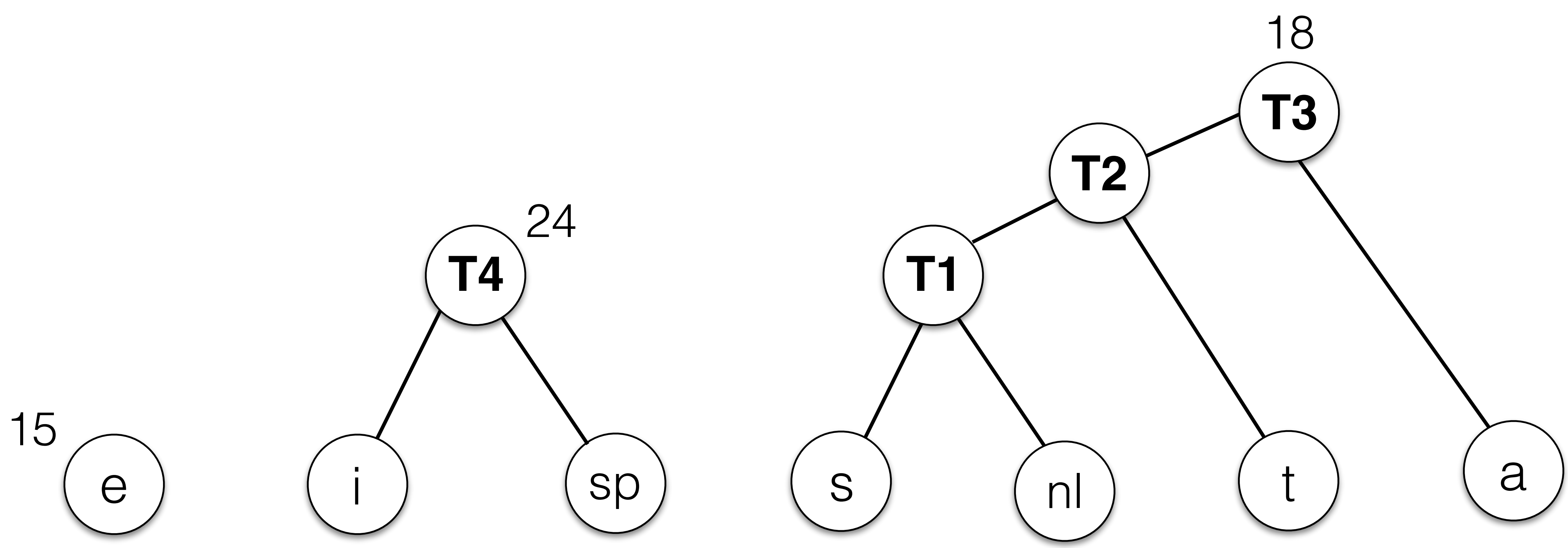
- In every phase:
  - Choose the two trees with smallest weight and merge them.

Huffman's Algorithm



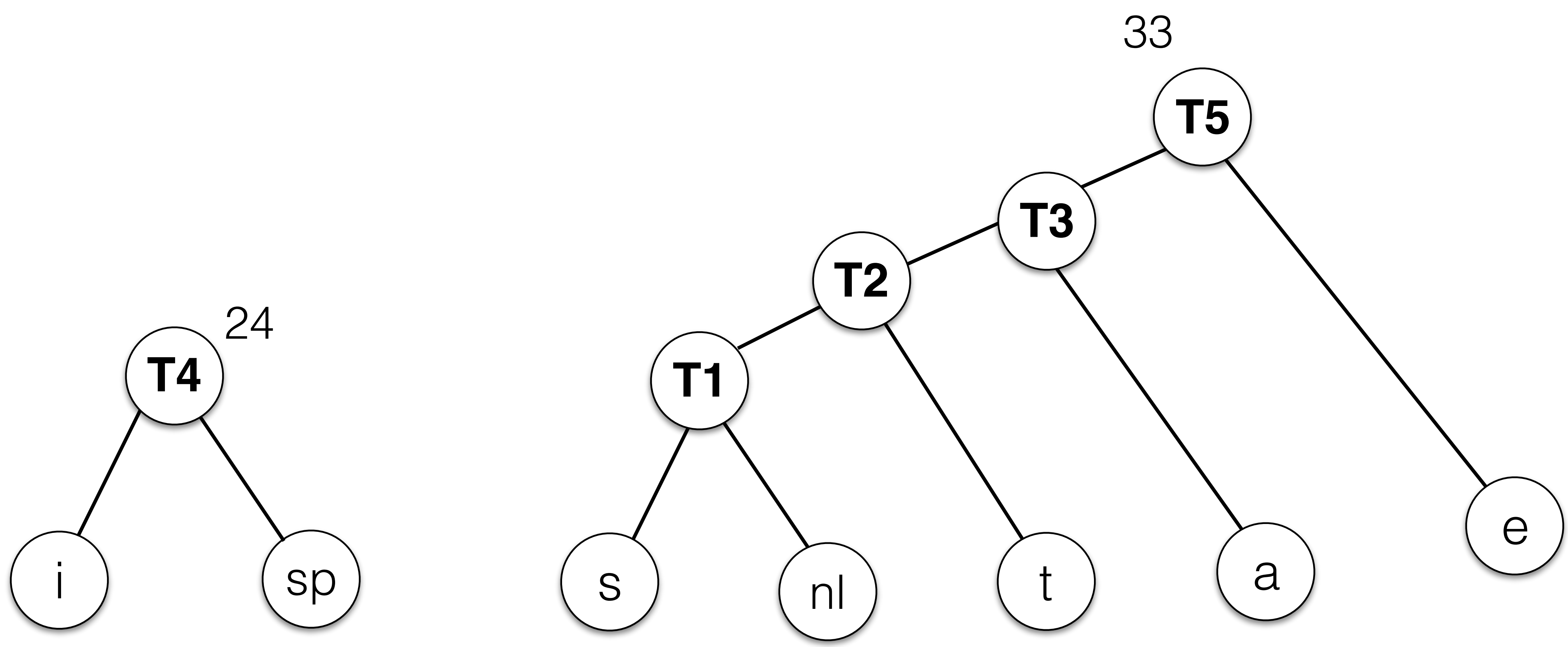
- In every phase:
  - Choose the two trees with smallest weight and merge them.

Huffman's Algorithm



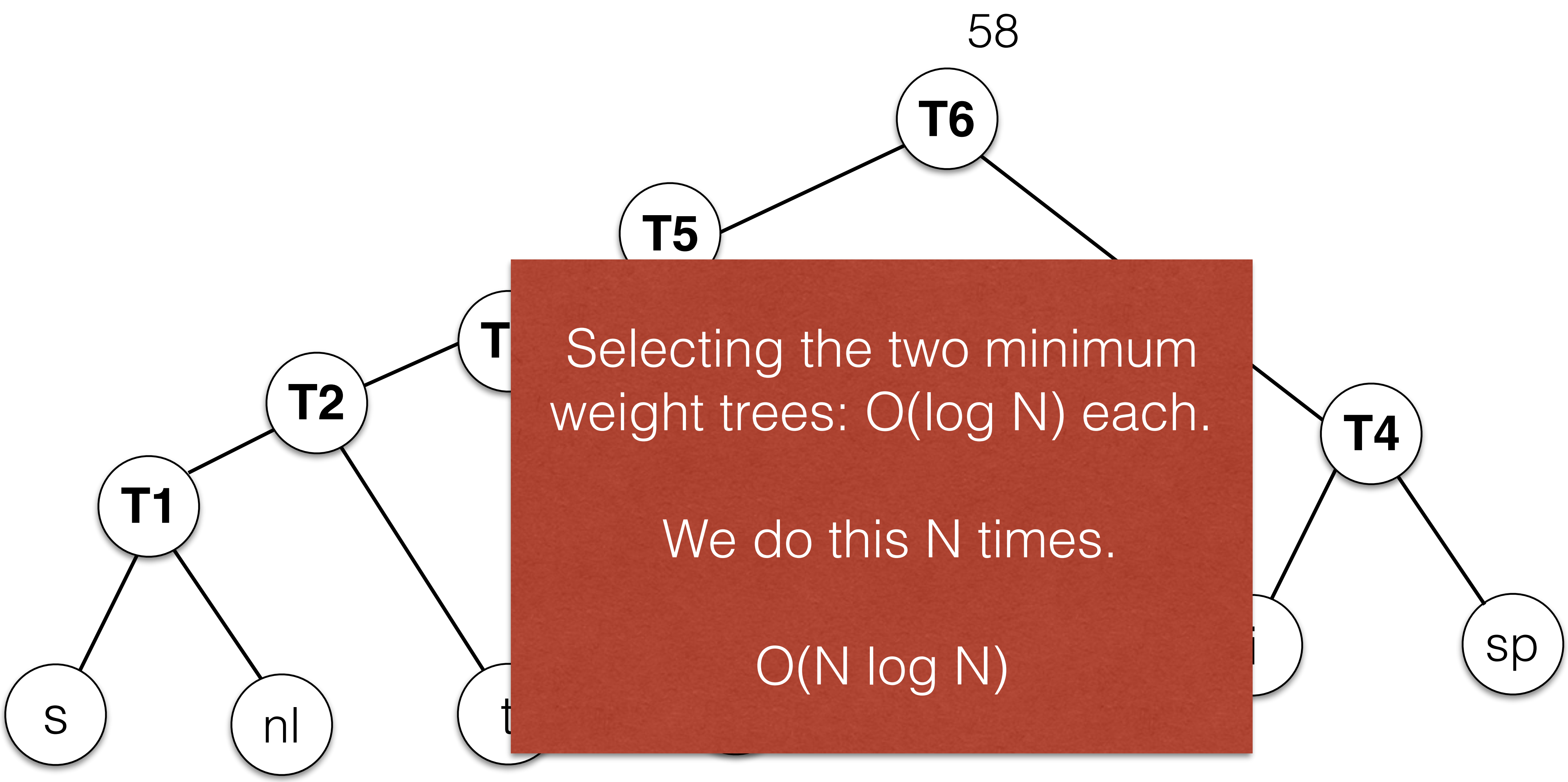
- In every phase:
  - Choose the two trees with smallest weight and merge them.

Huffman's Algorithm



- In every phase:
  - Choose the two trees with smallest weight and merge them.

Huffman's Algorithm



Selecting the two minimum weight trees: O(log N) each.

We do this N times.

O(N log N)

- This is clearly a greedy algorithm as we consider then two lowest-weight trees at any level.
- Keep the trees in the forest on a heap.

Divide and Conquer Algorithms

- Algorithms consist of two parts:
  - Divide: Decompose the problem into smaller sub-problems. Solve each problem recursively (down to the base case).
  - Conquer: Solve the problem by combining solutions to the sub-problem.

Divide and Conquer Example Algorithms

- Merge Sort. Quick Sort.
- Binary Search.
- Towers of Hanoi.
- These algorithms work efficiently because:
  - The subproblems are independent.
  - Solving the subproblems first makes the overall problem easier.

Merge Sort

- Split the array in half, recursively sort each half.
- Merge the two sorted lists.

34	8	64	2	51	32	21	1

Merge Sort

- Split the array in half, recursively sort each half.
- Merge the two sorted lists.

34	8	64	2

51	32	21	1

Merge Sort

- Split the array in half, recursively sort each half.
- Merge the two sorted lists.

34	8	64	2	51	32	21	1

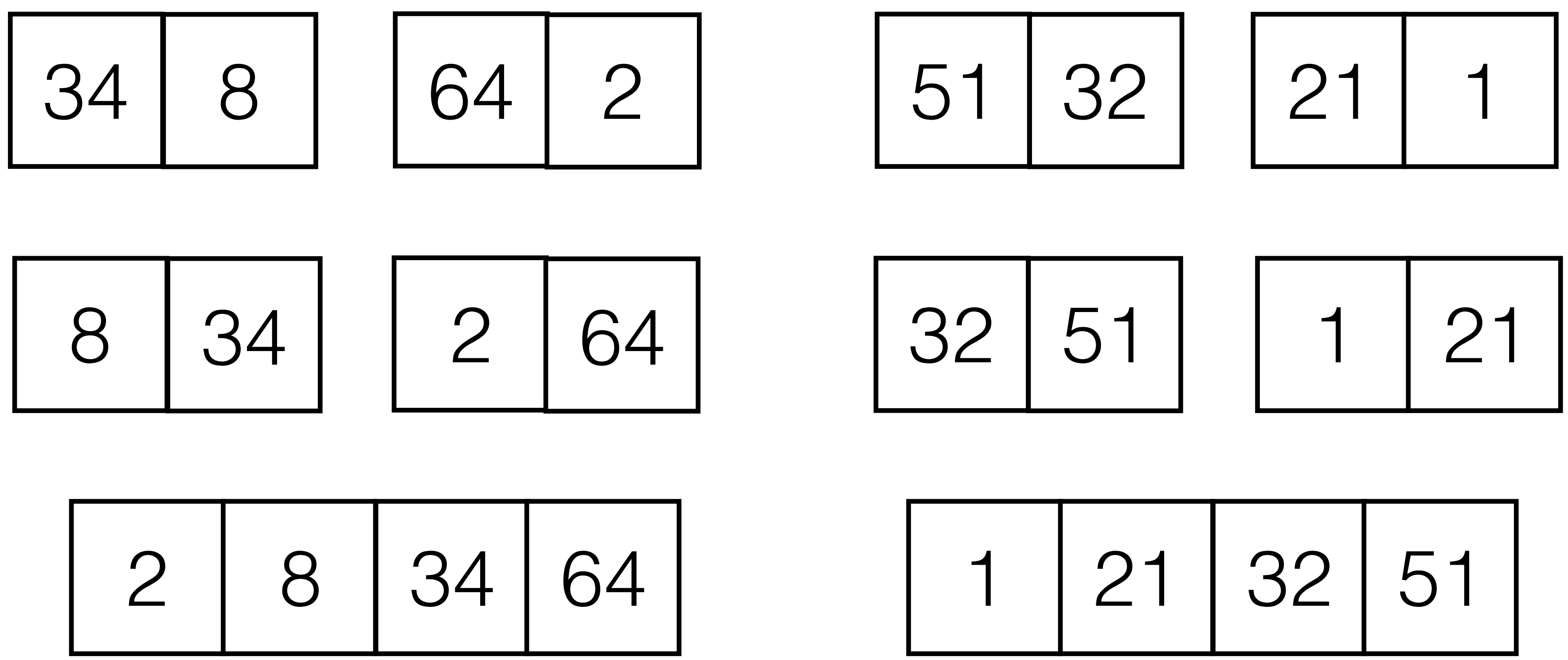
Merge Sort

- Split the array in half, recursively sort each half.
- Merge the two sorted lists.

34	8	64	2	51	32	21	1
8	34	2	64	32	51	1	21

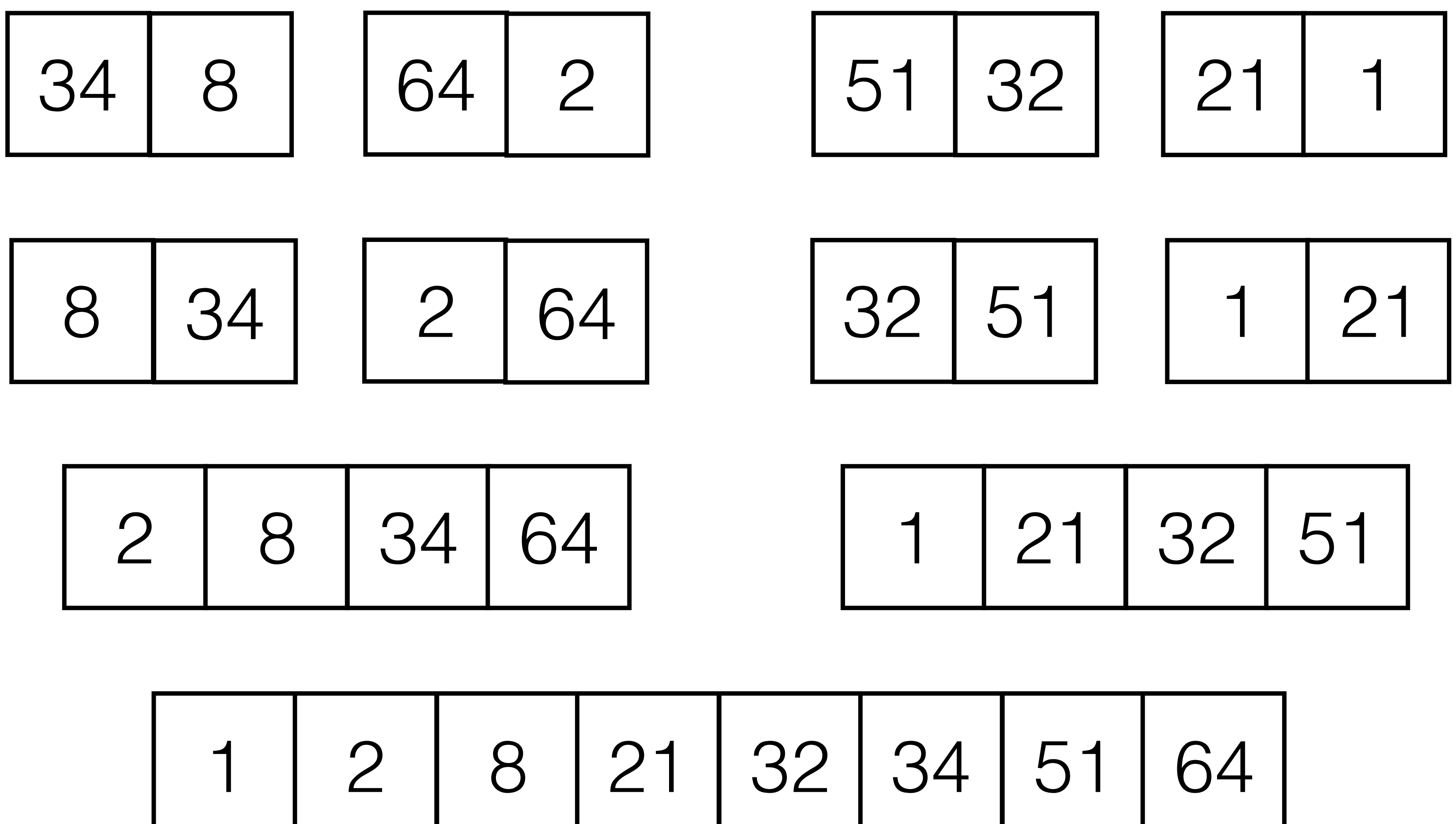
Merge Sort

- Split the array in half, recursively sort each half.
- Merge the two sorted lists.



Merge Sort

- Split the array in half, recursively sort each half.
- Merge the two sorted lists.



Merge Sort Running Time

- Base case: N=1 (sort a 1-element list). T(1) = 1
- Recurrence: T(N) = 2 T(N/2) + N

Merge the two halfs

Recursively sort each half

Running Time Analysis for Merge Sort and Quick Sort with Perfect Pivot.

$$T(N) = 2 \cdot T(\frac{N}{2}) + N$$

$$= 2 \cdot (2 \cdot T(\frac{N}{4}) + \frac{N}{2}) + N \qquad = 4 \cdot T(\frac{N}{4}) + N + N$$

$$= 2^k \cdot T(\frac{N}{2^k}) + k \cdot N$$

assume $k = \log N$

$$= N \cdot T(1) + logN \cdot N$$

$$= N + N \cdot \log N = \Theta(N \log N)$$

Running time of Divide and Conquer Algorithms: "Master Theorem"

Most divide and conquer algorithms have the following running time equation:

$$T(N) = aT(N/b) + \Theta(N^k)$$

The "Master Theorem" states that this recurrence relation has the following solution:

$$T(N) = \begin{cases} O(N^{\log_b a}) & \text{if } a > b^k \\ O(N^k \log N) & \text{if } a = b^k \\ O(N^k) & \text{if } a < b^k \end{cases}$$

Master Theorem: MergeSort

$$aT(N/b) + \Theta(N^k) = \begin{cases} O(N^{\log_b a}) & \text{if } a > b^k \\ O(N^k \log N) & \text{if } a = b^k \\ O(N^k) & \text{if } a < b^k \end{cases}$$

Example: Merge Sort $T(N) = 2 \cdot T(\frac{N}{2}) + N$

$$a = 2 \quad b = 2 \quad k = 1$$

This is Case 2: $T(N) = O(N \log N)$

Dynamic Programming Algorithms

- In some cases, recursive algorithms (such as the ones used for Divide and Conquer algorithms) won't work.
- That's because the solution to a subproblem is used more than once.
  - Merge Sort works because each partition is processed *exactly once.*
- Dynamic Programming algorithms solve this problem by systematically recording the solution to sub-problems in a table and re-using them later.

Broken Fibonacci

$$F_1 = 1, F_2 = 2$$

$$F_{k+1} = F_k + F_{k-1}$$

1,1,2,3,5,8,13,21,..

```
public int fibonacci(int k) throws IllegalArgumentException{
    if (k < 1) {
      throw new IllegalArgumentException("Expecting a positive integer.");
    }
    if (k == 1 | k == 2) {
      return 1;
    } else {
      return fibonacci(k-1) + fibonacci(k-2);
    }
}
```

Base case: 1 step T(1) = O(c), T(2) = O(c)

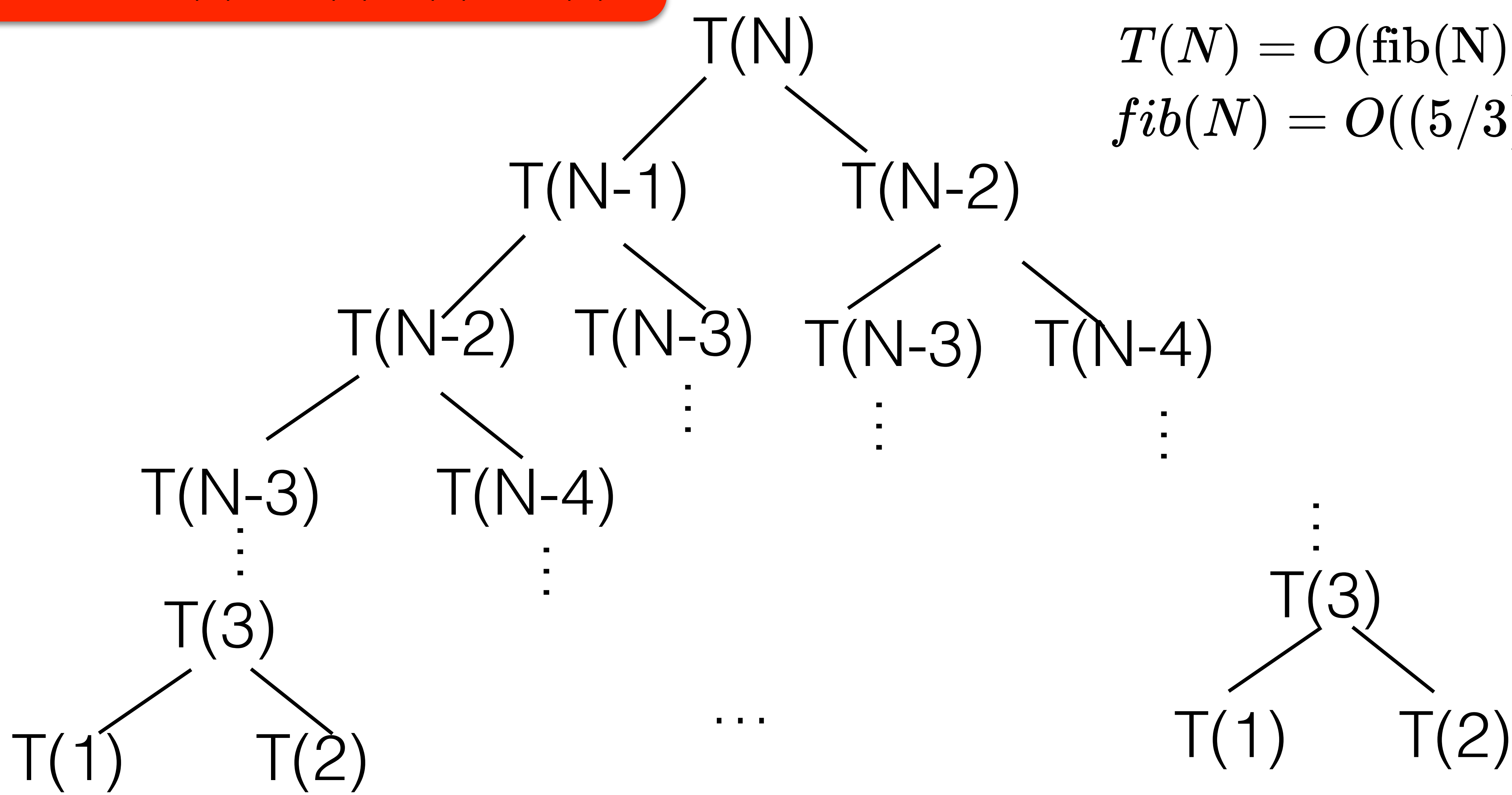
Recursive calls: T(k) = O(T(k-1) + T(k-2))

Analyzing the Recursive Fibonacci Solution

Recursive calls: T(k) = O(T(k-1) + T(k-2))
Base case: T(1) = O(c), T(2) = O(c)

each node is one recursive call

$$T(N) = O(\text{fib(N)})$$
$$fib(N) = O((5/3)^N)$$



Dynamic Programming
Fibonacci

```
public int fibonacci(int k) throws IllegalArgumentException{

  if (k < 1) {
    throw new IllegalArgumentException("Expecting a positive integer.");
  }
  int b = 1; //k-2
  int a = 1; //k-1
  for (int i=3; i<=k; i++) {
    int new_fib = a + b;
    b = a;
    a = new_fib;
  }
  return a;
}
```

$T(N) = O(N)$

Longest Increasing Subsequence

- Given a sequence of numbers, find the longest increasing (not necessarily contiguous) subsequence.

5 2 8 6 3 6 9 7

5 8 9

Longest Increasing Subsequence

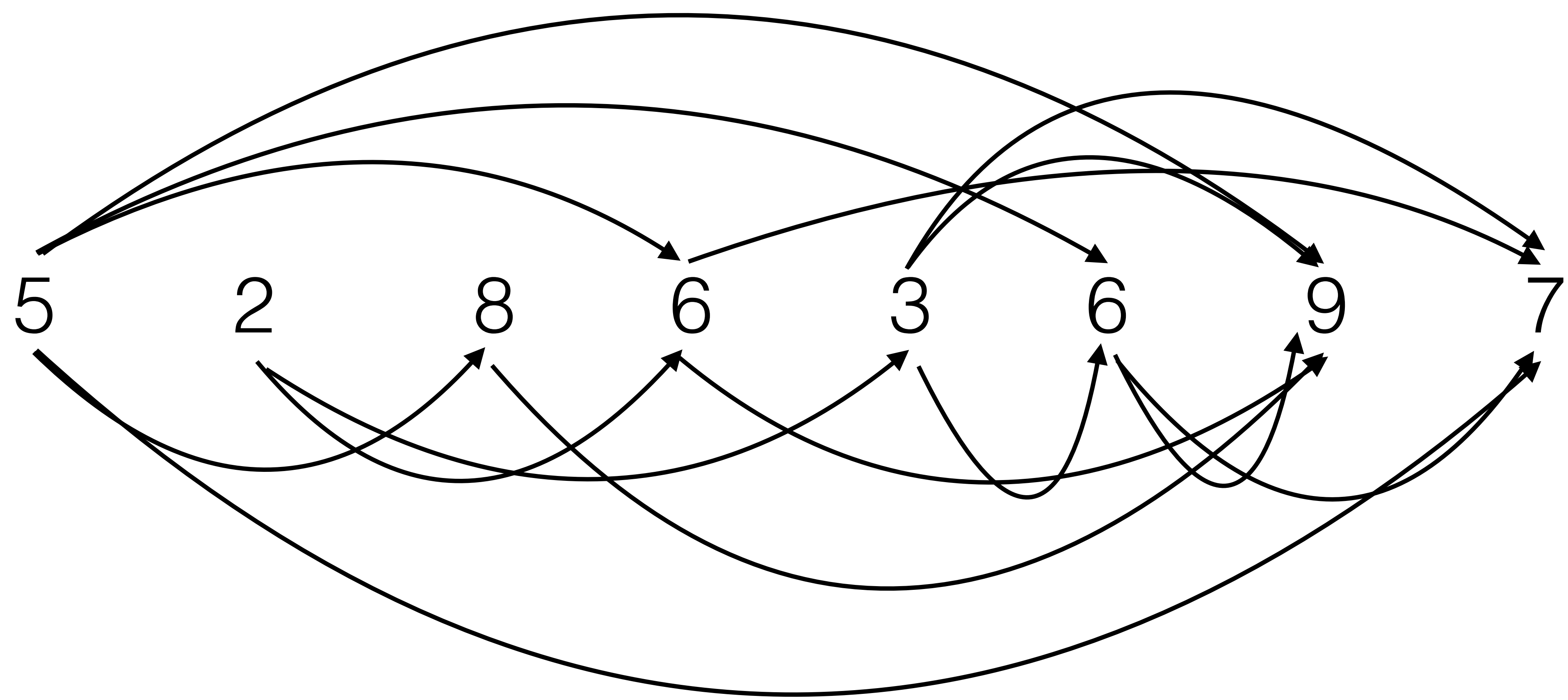
- Given a sequence of numbers, find the longest increasing (not necessarily contiguous) subsequence.

5 2 8 6 3 6 9 7

2 3 6 7

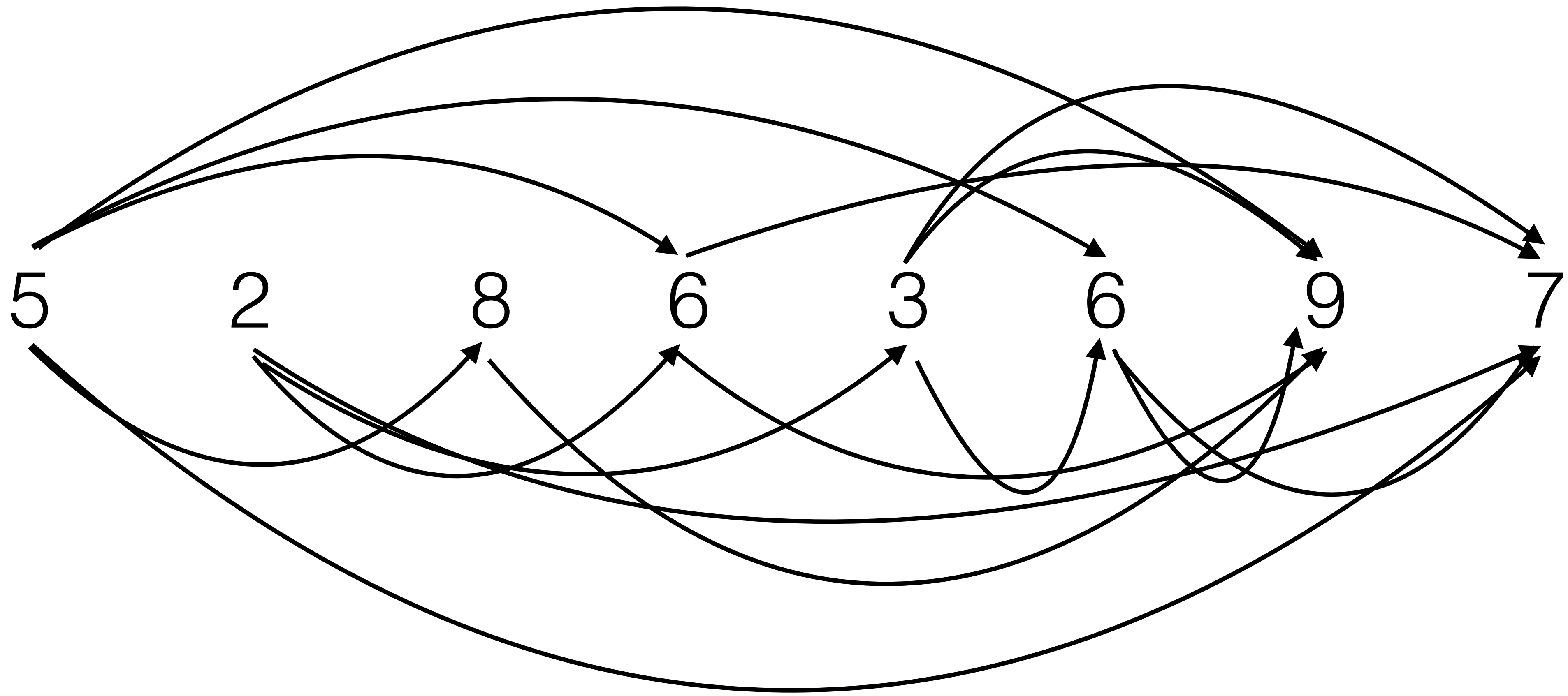
Longest Increasing Subsequence

- We can think of this problem as a graph problem.



This is a DAG. Our goal is to find the longest path.

Longest Increasing Subsequence: Recursive Solution

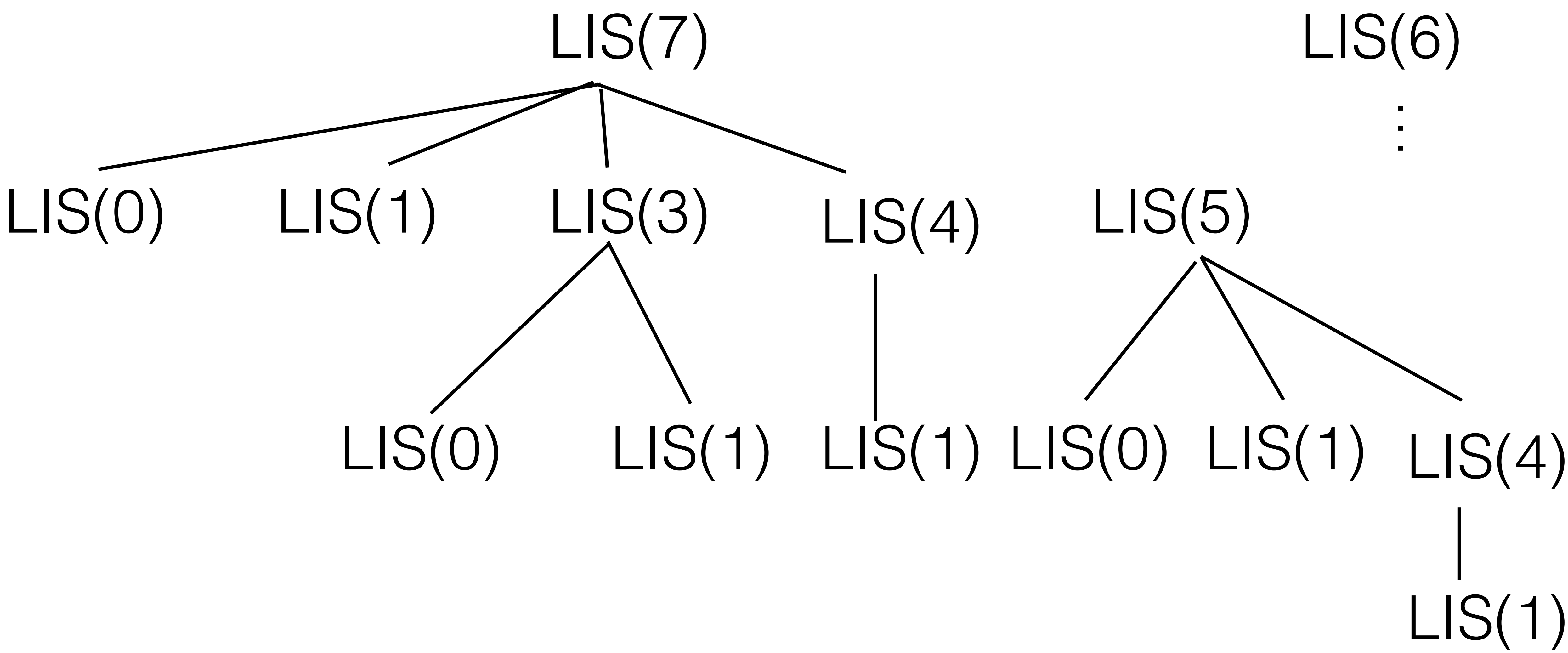


Step 1: Reducing the problem to easier subproblems (recursive divide-and-conquer solution)

```
LIS(i) {
    return max( {LIS(j) for j=j..i-1 if a[j] < a[i]} ) + 1
}
```

Longest Increasing Subsequence: Recursive Solution

```
LIS(i) {
    return max( {LIS(j) for j=0..i-1 if a[j] < a[i]} ) + 1
}
```



Longest Increasing Subsequence: Dynamic Programming

```
L = new Integer[n];
for i = 1…n {
    L[j] = max( {L(j) for j=0..i-1 if a[j] < a[i]} ) + 1
}
```

i	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
a[i]	5	2	8	6	3	6	9	7
L[i]	0	0						

Longest Increasing Subsequence: Dynamic Programming

```
L = new Integer[n];
for i = 1…n {
    L[j] = max( {L(j) for j=0..i-1 if a[j] < a[i]} ) + 1
}
```

i	0	1	2	3	4	5	6	7
a[i]	5	2	8	6	3	6	9	7
L[i]	0	0	1					

Longest Increasing Subsequence: Dynamic Programming

```
L = new Integer[n];
for i = 1…n {
    L[j] = max( {L(j) for j=0..i-1 if a[j] < a[i]} ) + 1
}
```

i	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
a[i]	5	2	8	6	3	6	9	7
L[i]	0	0	1	1				

Longest Increasing Subsequence: Dynamic Programming

```
L = new Integer[n];
for i = 1…n {
    L[j] = max( {L(j) for j=0..i-1 if a[j] < a[i]} ) + 1
}
```

i	0	1	2	3	4	5	6	7
a[i]	5	2	8	6	3	6	9	7
L[i]	0	0	1	1	1			

Longest Increasing Subsequence: Dynamic Programming

```
L = new Integer[n];
for i = 1…n {
    L[j] = max( {L(j) for j=0..i-1 if a[j] < a[i]} ) + 1
}
```

i	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
a[i]	5	2	8	6	3	6	9	7
L[i]	0	0	1	1	1	2		

Longest Increasing Subsequence: Dynamic Programming

```
L = new Integer[n];
for i = 1…n {
    L[j] = max( {L(j) for j=0..i-1 if a[j] < a[i]} ) + 1
}
```

i	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
a[i]	5	2	8	6	3	6	9	7
L[i]	0	0	1	1	1	2	3	

Longest Increasing Subsequence: Dynamic Programming

```
L = new Integer[n];
for i = 1…n {
    L[j] = max( {L(j) for j=0..i-1 if a[j] < a[i]} ) + 1
}
```

i	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
a[i]	5	2	8	6	3	6	9	7
L[i]	0	0	1	1	1	2	3	3

Longest Increasing Subsequence: Dynamic Programming

$O(N^2)$

```
L = new Integer[n];
for i = 1…n {
    L[j] = max( {L(j) for j=0..i-1 if a[j] < a[i]} ) + 1
}
```

i	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
a[i]	5	2	8	6	3	6	9	7
L[i]	0	0	1	1	1	2	3	3

Dynamic Programming Example: Minimum Edit Distance

- The Minimum *Edit Distance (Levenshtein Distance)* between two strings *s* and *t* is the minimal number of **insertions**, **deletions**, and **substitutions** needed to convert s into t.

SATURDAY

SUNDAY

Dynamic Programming Example: Minimum Edit Distance

- The Minimum *Edit Distance (Levenshtein Distance)* between two strings *s* and *t* is the minimal number of **insertions**, **deletions**, and **substitutions** needed to convert s into t.

SATURDAY

delete A

STURDAY

SUNDAY

Dynamic Programming Example: Minimum Edit Distance

- The Minimum *Edit Distance (Levenshtein Distance)* between two strings *s* and *t* is the minimal number of **insertions**, **deletions**, and **substitutions** needed to convert s into t.

SATURDAY

delete A

STURDAY

delete T

SURDAY

SUNDAY

Dynamic Programming Example: Minimum Edit Distance

- The Minimum *Edit Distance* between two strings *s* and *t* is the minimal number of **insertions**, **deletions**, and **substitutions** needed to convert s into t.

SATURDAY

delete A

STURDAY

Minimum Edit Distance = 3

delete T

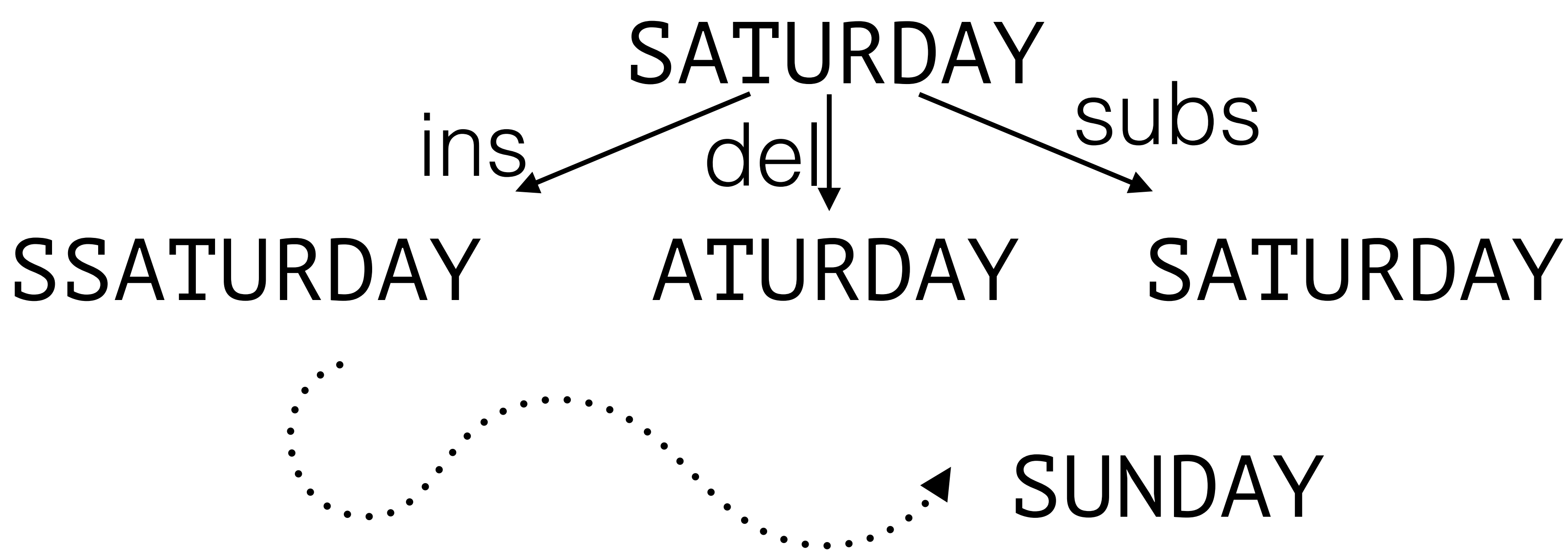
SURDAY

Substitute R with N

SUNDAY

Edit Distance as Search

- Initial state s, Goal state t.
- Try each operation for each letter.
- Try to find the shortest path.



- Search space is HUGE and contains **many duplicate states.**

Dynamic Programming Algorithm for Edit Distance

- Assume we have two strings
  $s = s_1, s_2, \ldots, s_n$ and $t = t_1, t_2, \ldots, t_m$

- Let D(i,j) be the minimum edit distance between s[0..i] and t[0..j].

- For example s = `SATURDAY`, t = `SUNDAY`
  D(2,3) = 2

$$\texttt{SA} \xrightarrow[\text{subs A / U}]{} \texttt{SU} \xrightarrow[\text{insert N}]{} \texttt{SUN}$$

Dynamic Programming Algorithm for Edit Distance

- Let D(i,j) be the minimum edit distance between s[0..i] and t[0..j].
- Basic approach:
  - Fill a table by computing D(i,j) for all ($0 < i < n$) and ($0 < j < m$).
  - Do this "bottom-up", starting with small i and j. Table entries for larger i and j are based on previous entries.

Dynamic Programming Algorithm for Edit Distance

For i = 1..n $D(i,0) = i$
For j = 1..m $D(0,j) = j$

For i = 1..n {
  For j = 1..m {

$$D(i,j) = min\begin{cases} D(i-1,j\qquad) + 1 \\ D(i\qquad,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 1 \text{ if } s_i \neq s_j \\ 0 \text{ if } s_i = s_j \end{cases} \end{cases}$$

  }
}

Edit Distance Example

initialization

$$D(i,0) = i$$

$$D(0,j) = j$$

D(i,j)							
Y	8						
A	7						
D	6						
R	5						
U	4						
T	3						
A	2						
S	1						
-	0	1	2	3	4	5	6
	-	**S**	**U**	**N**	**D**	**A**	**Y**

$$D(i,j) = min \begin{cases} D(i-1, j \quad\quad) + 1 & \text{deletion} \\ D(i \quad\quad , j-1) + 1 & \text{insertion} \\ D(i-1, j-1) + \begin{cases} 1 \text{ if } s_i \neq s_j \\ 0 \text{ if } s_i = s_j \end{cases} & \begin{matrix} \text{subst} \\ \text{or equal} \end{matrix} \end{cases}$$

D(i,j)							
Y	8						
A	7						
D	6						
R	5						
U	4						
T	3						
A	2						
S	1						
-	0	1	2	3	4	5	6
	-	**S**	**U**	**N**	**D**	**A**	**Y**

$$D(i,j) = min \begin{cases} D(i-1, j \quad\quad) + 1 & \text{deletion} \\ D(i \quad\quad, j-1) + 1 & \text{insertion} \\ D(i-1, j-1) + \begin{cases} 1 \text{ if } s_i \neq s_j \\ 0 \text{ if } s_i = s_j \end{cases} & \begin{matrix}\text{subst} \\ \text{or equal}\end{matrix} \end{cases}$$

D(i,j)							
Y	8						
A	7						
D	6						
R	5						
U	4						
T	3						
A	2						
S	**1**	0					
-	**0**	**1**	2	3	4	5	6
	-	**S**	**U**	**N**	**D**	**A**	**Y**

$$D(i,j) = min \begin{cases} D(i-1, j) + 1 & \text{deletion} \\ D(i, j-1) + 1 & \text{insertion} \\ D(i-1, j-1) + \begin{cases} 1 \text{ if } s_i \neq s_j & \text{subst} \\ 0 \text{ if } s_i = s_j & \text{or equal} \end{cases} \end{cases}$$

D(i,j)							
Y	8						
A	7						
D	6						
R	5						
U	4						
T	3						
A	2						
S	1	**0**	1				
-	0	**1**	**2**	3	4	5	6
	-	**S**	**U**	**N**	**D**	**A**	**Y**

$$D(i,j) = min \begin{cases} D(i-1, j) + 1 & \text{deletion} \\ D(i, j-1) + 1 & \text{insertion} \\ D(i-1, j-1) + \begin{cases} 1 \text{ if } s_i \neq s_j \\ 0 \text{ if } s_i = s_j \end{cases} & \begin{matrix}\text{subst} \\ \text{or equal}\end{matrix} \end{cases}$$

D(i,j)							
Y	8						
A	7						
D	6						
R	5						
U	4						
T	3						
A	2						
S	1	0	**1**	2			
-	0	1	**2**	**3**	4	5	6
	-	**S**	**U**	**N**	**D**	**A**	**Y**

$$D(i,j) = min \begin{cases} D(i-1, j \quad\quad) + 1 & \text{deletion} \\ D(i \quad\quad , j-1) + 1 & \text{insertion} \\ D(i-1, j-1) + \begin{cases} 1 \text{ if } s_i \neq s_j & \text{subst} \\ 0 \text{ if } s_i = s_j & \text{or equal} \end{cases} \end{cases}$$

D(i,j)							
Y	8						
A	7						
D	6						
R	5						
U	4						
T	3						
A	2						
S	1	0	1	**2**	3		
-	0	1	2	**3**	**4**	5	6
	-	**S**	**U**	**N**	**D**	**A**	**Y**

$$D(i,j) = min \begin{cases} D(i-1, j \quad\quad) + 1 & \text{deletion} \\ D(i \quad\quad , j-1) + 1 & \text{insertion} \\ D(i-1, j-1) + \begin{cases} 1 \text{ if } s_i \neq s_j \\ 0 \text{ if } s_i = s_j \end{cases} & \begin{matrix}\text{subst} \\ \text{or equal}\end{matrix} \end{cases}$$

D(i,j)							
Y	8						
A	7						
D	6						
R	5						
U	4						
T	3						
A	2						
S	1	0	1	2	**3**	4	
-	0	1	2	3	**4**	**5**	6
	-	**S**	**U**	**N**	**D**	**A**	**Y**

$$D(i,j) = min \begin{cases} D(i-1, j \quad\quad) + 1 & \text{deletion} \\ D(i \quad\quad , j-1) + 1 & \text{insertion} \\ D(i-1, j-1) + \begin{cases} 1 \text{ if } s_i \neq s_j \\ 0 \text{ if } s_i = s_j \end{cases} & \begin{matrix} \text{subst} \\ \text{or equal} \end{matrix} \end{cases}$$

D(i,j)							
Y	8						
A	7						
D	6						
R	5						
U	4						
T	3						
A	2						
S	1	0	1	2	3	**4**	5
-	0	1	2	3	4	**5**	**6**
	-	**S**	**U**	**N**	**D**	**A**	**Y**

$$D(i,j) = min \begin{cases} D(i-1, j) + 1 & \text{deletion} \\ D(i, j-1) + 1 & \text{insertion} \\ D(i-1, j-1) + \begin{cases} 1 \text{ if } s_i \neq s_j \\ 0 \text{ if } s_i = s_j \end{cases} & \begin{array}{l}\text{subst} \\ \text{or equal}\end{array} \end{cases}$$

D(i,j)							
Y	8						
A	7						
D	6						
R	5						
U	4						
T	3						
A	**2**	1					
S	**1**	**0**	1	2	3	4	5
-	0	1	2	3	4	5	6
	-	**S**	**U**	**N**	**D**	**A**	**Y**

$$D(i,j) = min \begin{cases} D(i-1, j \quad\quad) + 1 & \text{deletion} \\ D(i \quad\quad , j-1) + 1 & \text{insertion} \\ D(i-1, j-1) + \begin{cases} 1 \text{ if } s_i \neq s_j \\ 0 \text{ if } s_i = s_j \end{cases} & \text{subst or equal} \end{cases}$$

D(i,j)							
Y	8						
A	7						
D	6						
R	5						
U	4						
T	3						
A	2	**1**	1				
S	1	**0**	**1**	2	3	4	5
-	0	1	2	3	4	5	6
	-	**S**	**U**	**N**	**D**	**A**	**Y**

$$D(i,j) = min \begin{cases} D(i-1, j \quad\quad) + 1 & \text{deletion} \\ D(i \quad\quad, j-1) + 1 & \text{insertion} \\ D(i-1, j-1) + \begin{cases} 1 \text{ if } s_i \neq s_j \\ 0 \text{ if } s_i = s_j \end{cases} & \text{subst or equal} \end{cases}$$

D(i,j)							
Y	8						
A	7						
D	6						
R	5						
U	4						
T	3						
A	2	1	**1**	2			
S	1	0	**1**	**2**	3	4	5
-	0	1	2	3	4	5	6
	-	**S**	**U**	**N**	**D**	**A**	**Y**

$$D(i,j) = min\begin{cases} D(i-1, j \quad\quad) + 1 & \text{deletion} \\ D(i \quad\quad , j-1) + 1 & \text{insertion} \\ D(i-1, j-1) + \begin{cases} 1 \text{ if } s_i \neq s_j \\ 0 \text{ if } s_i = s_j \end{cases} & \begin{matrix}\text{subst} \\ \text{or equal}\end{matrix} \end{cases}$$

D(i,j)							
Y	8						
A	7						
D	6						
R	5						
U	4						
T	3						
A	2	1	1	**2**	3		
S	1	0	1	**2**	**3**	4	5
-	0	1	2	3	4	5	6
	-	**S**	**U**	**N**	**D**	**A**	**Y**

$$D(i,j) = min \begin{cases} D(i-1, j\quad\quad) + 1 & \text{deletion} \\ D(i\quad\quad, j-1) + 1 & \text{insertion} \\ D(i-1, j-1) + \begin{cases} 1 \text{ if } s_i \neq s_j \\ 0 \text{ if } s_i = s_j \end{cases} & \begin{matrix}\text{subst} \\ \text{or equal}\end{matrix} \end{cases}$$

D(i,j)							
Y	8						
A	7						
D	6						
R	5						
U	4						
T	3						
A	2	1	1	2	**3**	3	
S	1	0	1	2	**3**	**4**	5
-	0	1	2	3	4	5	6
	-	**S**	**U**	**N**	**D**	**A**	**Y**

$$D(i,j) = min \begin{cases} D(i-1, j \quad\quad) + 1 & \text{deletion} \\ D(i \quad\quad , j-1) + 1 & \text{insertion} \\ D(i-1, j-1) + \begin{cases} 1 \text{ if } s_i \neq s_j & \text{subst} \\ 0 \text{ if } s_i = s_j & \text{or equal} \end{cases} \end{cases}$$

D(i,j)							
Y	8						
A	7						
D	6						
R	5						
U	4						
T	3						
A	2	1	1	2	3	**3**	4
S	1	0	1	2	3	**4**	**5**
-	0	1	2	3	4	5	6
	-	**S**	**U**	**N**	**D**	**A**	**Y**

$$D(i,j) = min \begin{cases} D(i-1, j \qquad) + 1 & \text{deletion} \\ D(i \qquad, j-1) + 1 & \text{insertion} \\ D(i-1, j-1) + \begin{cases} 1 \text{ if } s_i \neq s_j \\ 0 \text{ if } s_i = s_j \end{cases} & \begin{matrix}\text{subst} \\ \text{or equal}\end{matrix} \end{cases}$$

D(i,j)							
Y	8						
A	7						
D	6						
R	5						
U	4						
T	3	2	2	2	3	4	4
A	2	1	1	2	3	3	4
S	1	0	1	2	3	4	5
-	0	1	2	3	4	5	6
	-	**S**	**U**	**N**	**D**	**A**	**Y**

$$D(i,j) = min \begin{cases} D(i-1, j\quad\ ) + 1 & \text{deletion} \\ D(i\quad\ , j-1) + 1 & \text{insertion} \\ D(i-1, j-1) + \begin{cases} 1 \text{ if } s_i \neq s_j \\ 0 \text{ if } s_i = s_j \end{cases} & \begin{matrix}\text{subst} \\ \text{or equal}\end{matrix} \end{cases}$$

D(i,j)							
Y	8						
A	7						
D	6						
R	5						
U	4	3	2	3	3	4	5
T	3	2	2	2	3	4	4
A	2	1	1	2	3	3	4
S	1	0	1	2	3	4	5
-	0	1	2	3	4	5	6
	-	**S**	**U**	**N**	**D**	**A**	**Y**

$$D(i,j) = min\begin{cases} D(i-1,j\quad) + 1 & \text{deletion} \\ D(i\quad, j-1) + 1 & \text{insertion} \\ D(i-1,j-1) + \begin{cases} 1 \text{ if } s_i \neq s_j \\ 0 \text{ if } s_i = s_j \end{cases} & \text{subst or equal} \end{cases}$$

D(i,j)							
Y	8						
A	7						
D	6						
R	5	4	3	3	4	4	5
U	4	3	2	3	3	4	5
T	3	2	2	2	3	4	4
A	2	1	1	2	3	3	4
S	1	0	1	2	3	4	5
-	0	1	2	3	4	5	6
	-	**S**	**U**	**N**	**D**	**A**	**Y**

$$D(i,j) = min \begin{cases} D(i-1, j \quad\quad) + 1 & \text{deletion} \\ D(i \quad\quad , j-1) + 1 & \text{insertion} \\ D(i-1, j-1) + \begin{cases} 1 \text{ if } s_i \neq s_j \\ 0 \text{ if } s_i = s_j \end{cases} & \begin{matrix}\text{subst} \\ \text{or equal}\end{matrix} \end{cases}$$

D(i,j)							
Y	8						
A	7						
D	6	5	4	4	3	4	5
R	5	4	3	3	4	4	5
U	4	3	2	3	3	4	5
T	3	2	2	2	3	4	4
A	2	1	1	2	3	3	4
S	1	0	1	2	3	4	5
-	0	1	2	3	4	5	6
	-	**S**	**U**	**N**	**D**	**A**	**Y**

$$D(i,j) = min\begin{cases} D(i-1, j\quad\quad) + 1 & \text{deletion} \\ D(i\quad\quad, j-1) + 1 & \text{insertion} \\ D(i-1, j-1) + \begin{cases} 1 \text{ if } s_i \neq s_j \\ 0 \text{ if } s_i = s_j \end{cases} & \begin{matrix}\text{subst} \\ \text{or equal}\end{matrix} \end{cases}$$

D(i,j)							
Y	8						
A	7	6	5	5	4	3	4
D	6	5	4	4	3	4	5
R	5	4	3	3	4	4	5
U	4	3	2	3	3	4	5
T	3	2	2	2	3	4	4
A	2	1	1	2	3	3	4
S	1	0	1	2	3	4	5
-	0	1	2	3	4	5	6
	-	**S**	**U**	**N**	**D**	**A**	**Y**

$$D(i,j) = min \begin{cases} D(i-1, j \quad\quad) + 1 & \text{deletion} \\ D(i \quad\quad , j-1) + 1 & \text{insertion} \\ D(i-1, j-1) + \begin{cases} 1 \text{ if } s_i \neq s_j \\ 0 \text{ if } s_i = s_j \end{cases} & \begin{matrix}\text{subst} \\ \text{or equal}\end{matrix} \end{cases}$$

D(i,j)							
Y	8	7	6	6	5	4	3
A	7	6	5	5	4	3	4
D	6	5	4	4	3	4	5
R	5	4	3	3	4	4	5
U	4	3	2	3	3	4	5
T	3	2	2	2	3	4	4
A	2	1	1	2	3	3	4
S	1	0	1	2	3	4	5
-	0	1	2	3	4	5	6
	-	**S**	**U**	**N**	**D**	**A**	**Y**

$$
D(i,j) = min\begin{cases} D(i-1, j \quad\quad) + 1 & \text{deletion} \\ D(i \quad\quad , j-1) + 1 & \text{insertion} \\ D(i-1, j-1) + \begin{cases} 1 \text{ if } s_i \neq s_j & \text{subst} \\ 0 \text{ if } s_i = s_j & \text{or equal} \end{cases} \end{cases}
$$

D(i,j)							
Y	8	7	6	6	5	4	3
A	7	6	5	5	4	3	4
D	6	5	4	4	3	4	5
R	5	4	3 subs	3	4	4	5
U	4	3	2	3	3	4	5
T	3	2	2	2	3	4	4
A	2	1	1	2	3	3	4
S	1	0	1	2	3	4	5
-	0	1	2	3	4	5	6
	-	**S**	**U**	**N**	**D**	**A**	**Y**

$$D(i,j) = min\begin{cases} D(i-1, j\quad\quad) + 1 & \text{deletion} \\ D(i\quad\quad, j-1) + 1 & \text{insertion} \\ D(i-1, j-1) + \begin{cases} 1 \text{ if } s_i \neq s_j \\ 0 \text{ if } s_i = s_j \end{cases} & \begin{matrix}\text{subst} \\ \text{or equal}\end{matrix} \end{cases}$$

D(i,j)							
Y	8	7	6	6	5	4	3
A	7	6	5	5	4	3	4
D	6	5	4	4	3	4	5
R	5	4	3 subs	3	4	4	5
U	4	3	2	3	3	4	5
T	3	2	2	2	3	4	4
A	2	1	1	2	3	3	4
S	1	0	1	2	3	4	5
-	0	1	2	3	4	5	6
	-	**S**	**U**	**N**	**D**	**A**	**Y**

$$D(i,j) = min \begin{cases} D(i-1,j) + 1 & \text{deletion} \\ D(i,j-1) + 1 & \text{insertion} \\ D(i-1,j-1) + \begin{cases} 1 \text{ if } s_i \neq s_j \\ 0 \text{ if } s_i = s_j \end{cases} & \begin{matrix}\text{subst} \\ \text{or equal}\end{matrix} \end{cases}$$

D(i,j)							
Y	8	7	6	6	5	4	3
A	7	6	5	5	4	3	4
D	6	5	4	4	3	4	5
R	5	4	3 subs	3	4	4	5
U	4	3	2	3	3	4	5
T	3	2	2 ins	2	3	4	4
A	2	1	1 ins	2	3	3	4
S	1	0	1	2	3	4	5
-	0	1	2	3	4	5	6
	-	**S**	**U**	**N**	**D**	**A**	**Y**

$$D(i,j) = min \begin{cases} D(i-1, j) + 1 & \text{deletion} \\ D(i, j-1) + 1 & \text{insertion} \\ D(i-1, j-1) + \begin{cases} 1 \text{ if } s_i \neq s_j \\ 0 \text{ if } s_i = s_j \end{cases} & \text{subst or equal} \end{cases}$$

SUNDAY

$\downarrow$ +A

SAUNDAY

$\downarrow$ + T

SATUNDAY

$\downarrow$ N/R

SATURDAY

D(i,j)							
Y	8	7	6	6	5	4	3
A	7	6	5	5	4	3	4
D	6	5	4	4	3	4	5
R	5	4	3 subs	3	4	4	5
U	4	3	2	3	3	4	5
T	3	2	2	2	3	4	4
A	2	1 ins	1	2	3	3	4
S	1	0 ins	1	2	3	4	5
-	0	1	2	3	4	5	6
	-	**S**	**U**	**N**	**D**	**A**	**Y**