CS141: Intermediate Data Structures and Algorithms

Greedy Algorithms

Yan Gu

Thanks to Ahmed Eldawy for sharing the slides

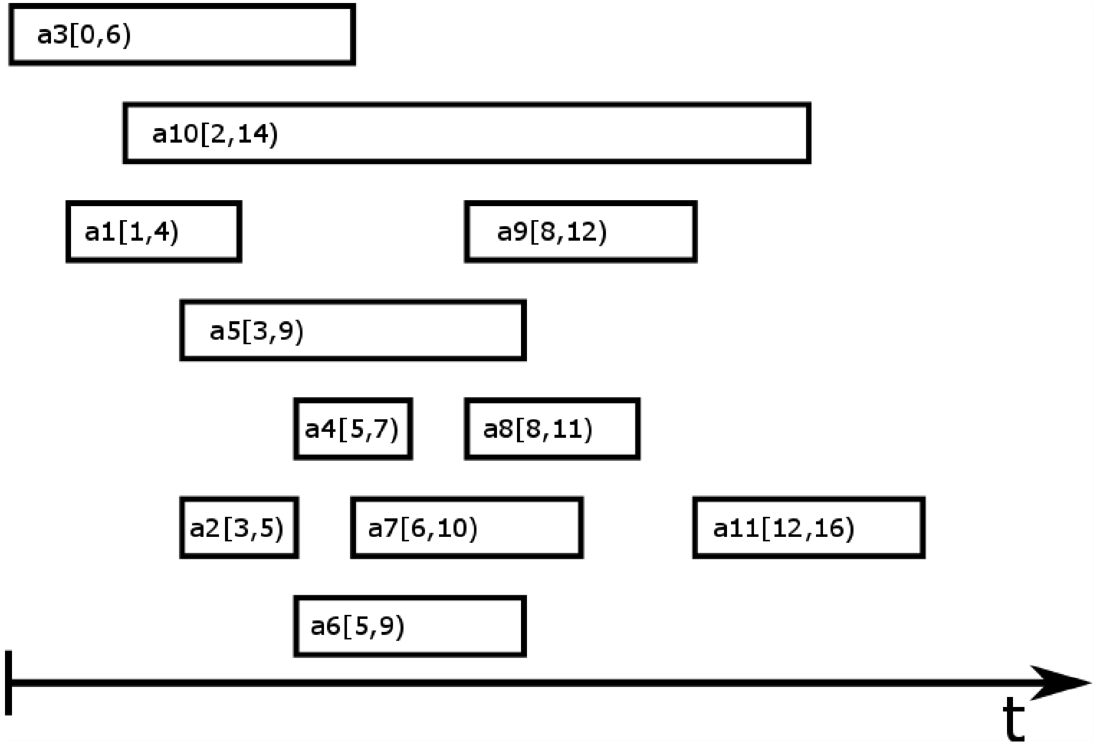
Greedy algorithm

- **Optimization problem:**
  - Find a **set** (or a **sequence**) of “**items**”
  - That satisfy some constraints and simultaneously optimize (i.e., **maximize** or **minimize**) some **objective function**

- **Greedy strategy**
  - Adds items to the solution one-by-one
  - Always choose the current best solution
  - No backtracking

Activity selection (task scheduling)

- **Given a set of activities $S = \{a_1, a_2, \dots, a_n\}$ where**
  - Each activity i has a start time s_i and a finish time f_i, where $0 \leq s_i < f_i < \infty$.
  - An activity a_i happens in the half-open time interval $[s_i, f_i)$.
  - Two activities are said to be **compatible** if they **do not overlap**.



- **The problem is to find a maximum-size compatible subset, i.e., a one with the maximum number of activities.**

Solution: earliest finish first!

Always choose the one that finishes earliest

Prove the optimality of a greedy algorithm: activity selection

1. **Greedy Choice: The greedy choice is part of the answer**
   - The earliest finish activity $\boldsymbol{a_m}$ is part of some optimal solution

2. **Optimal Substructure: The optimal solution to the big problem contains the optimal solution to the sub-problem**
   - Optimal solution $\{a_i, \dots\}$ without a_i is "the best solution of S − {those incompatible with a_i}"
   - Best solution with a_i is $\{a_i\} \cup$ "the best solution of S − {those incompatible with a_i}"

Huffman Tree and Huffman Codes

Merge pebbles

- **We have piles of pebbles:**





12 7 8 15 4

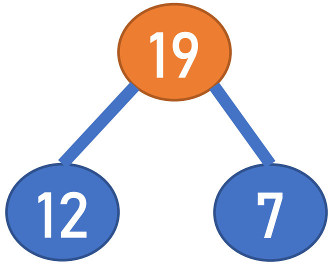
- **We want to merge them into one pile, but**
  - We can only merge two of them at a time
  - Merging two piles of size a and b cost you a+b units of energy (Let's assume you need to move both piles)
  - (e.g., merging 12 and 7 results in a new pile of size (12+7=)19, and cost you 19 units of energy
- **How can we merge all of them with the least energy?**

Merge pebbles

- **We have pebble piles and want to merge them into one pile, but**
  - We can only merge two of them at a time
  - Merging two piles of size a and b cost you a+b units of energy
- **Use a tree to represent the trace of merging**

Merge pebbles

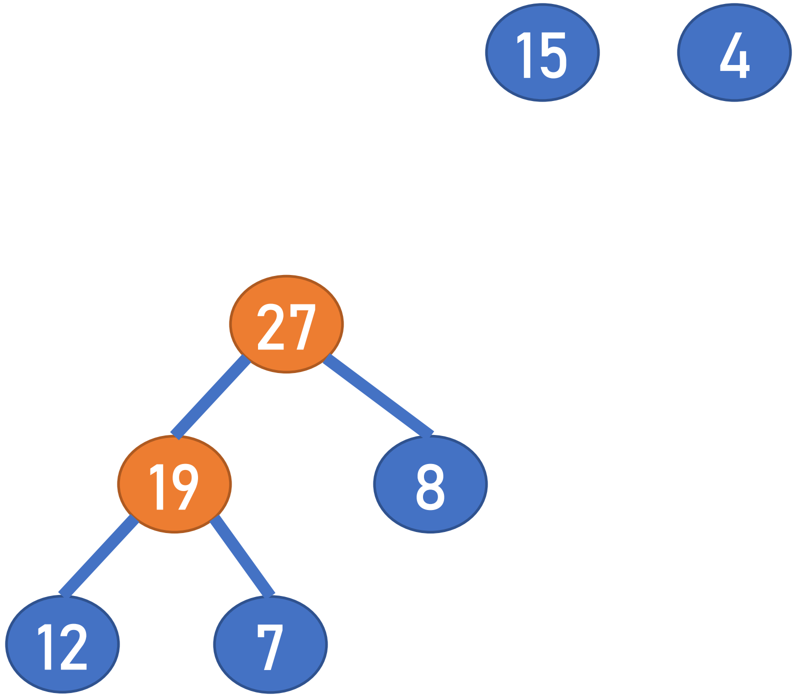
- **We have pebble piles and want to merge them into one pile, but**
  - We can only merge two of them at a time
  - Merging two piles of size a and b cost you a+b units of energy
- **Use a tree to represent the trace of merging**



Energy cost: 19

Merge pebbles

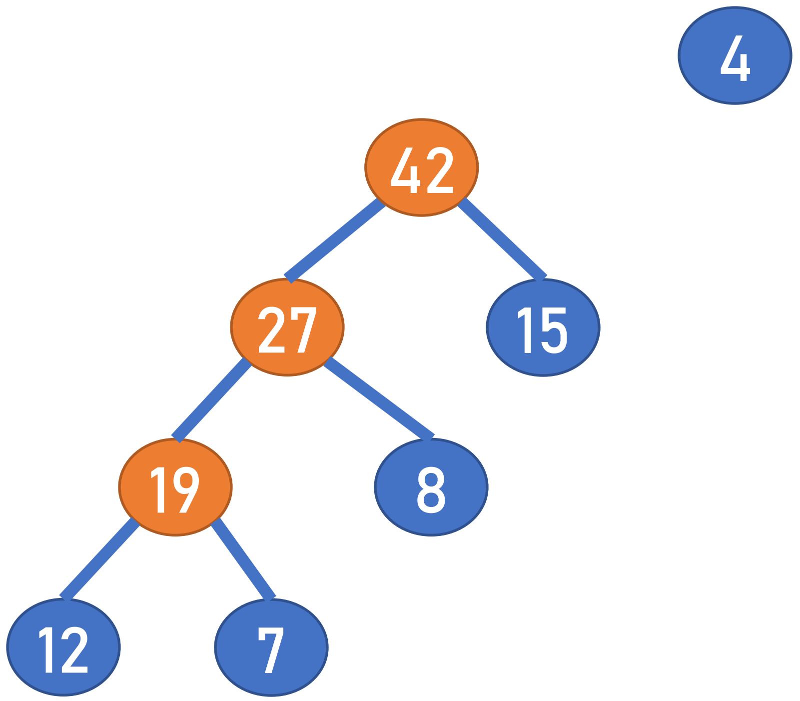
- **We have pebble piles and want to merge them into one pile, but**
  - We can only merge two of them at a time
  - Merging two piles of size a and b cost you a+b units of energy
- **Use a tree to represent the trace of merging**



Energy cost: 19 +27

Merge pebbles

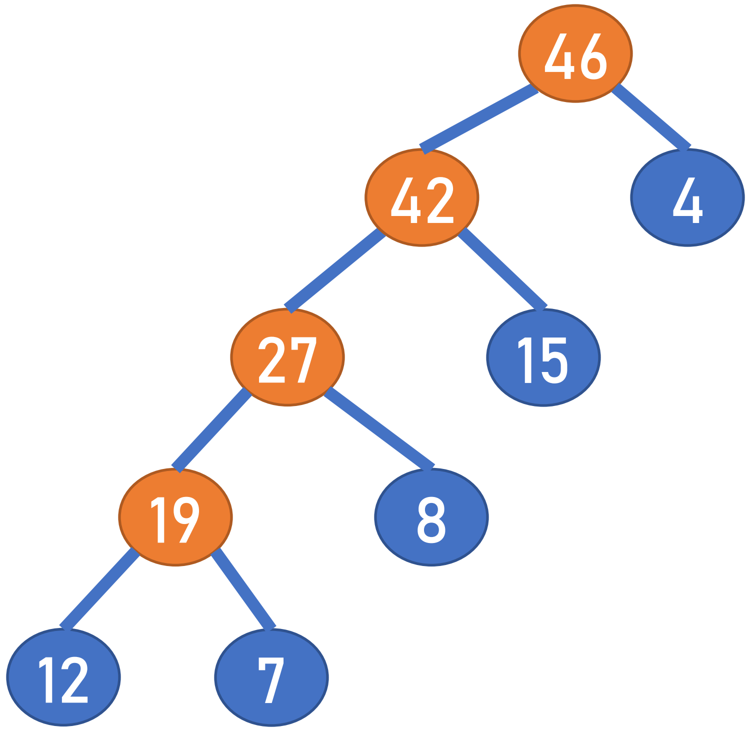
- **We have pebble piles and want to merge them into one pile, but**
  - We can only merge two of them at a time
  - Merging two piles of size a and b cost you a+b units of energy
- **Use a tree to represent the trace of merging**



Energy cost: 19 +27 +42

Merge pebbles

- **We have pebble piles and want to merge them into one pile, but**
  - We can only merge two of them at a time
  - Merging two piles of size a and b cost you a+b units of energy
- **Use a tree to represent the trace of merging**



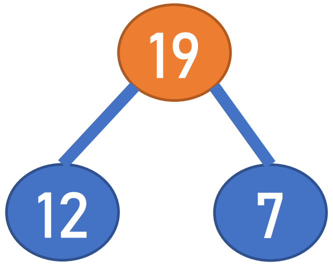
Energy cost: 19 +27 +42 +46 =134

Merge pebbles – another solution

- **We have pebble piles and want to merge them into one pile, but**
  - We can only merge two of them at a time
  - Merging two piles of size a and b cost you a+b units of energy
- **Use a tree to represent the trace of merging**

Merge pebbles – another solution

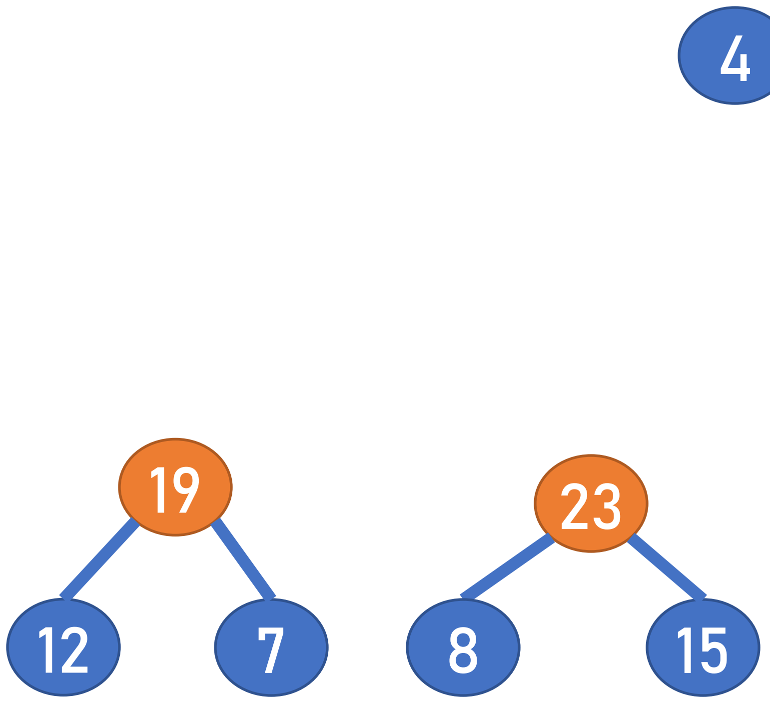
- **We have pebble piles and want to merge them into one pile, but**
  - We can only merge two of them at a time
  - Merging two piles of size a and b cost you a+b units of energy
- **Use a tree to represent the trace of merging**



Energy cost: 19

Merge pebbles – another solution

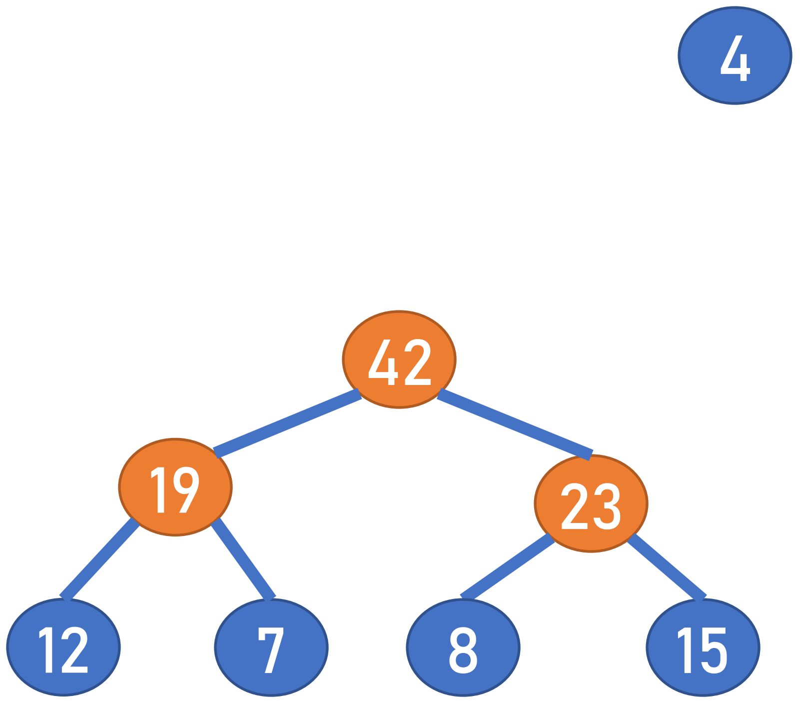
- **We have pebble piles and want to merge them into one pile, but**
  - We can only merge two of them at a time
  - Merging two piles of size a and b cost you a+b units of energy
- **Use a tree to represent the trace of merging**



Energy cost: 19 +23

Merge pebbles – another solution

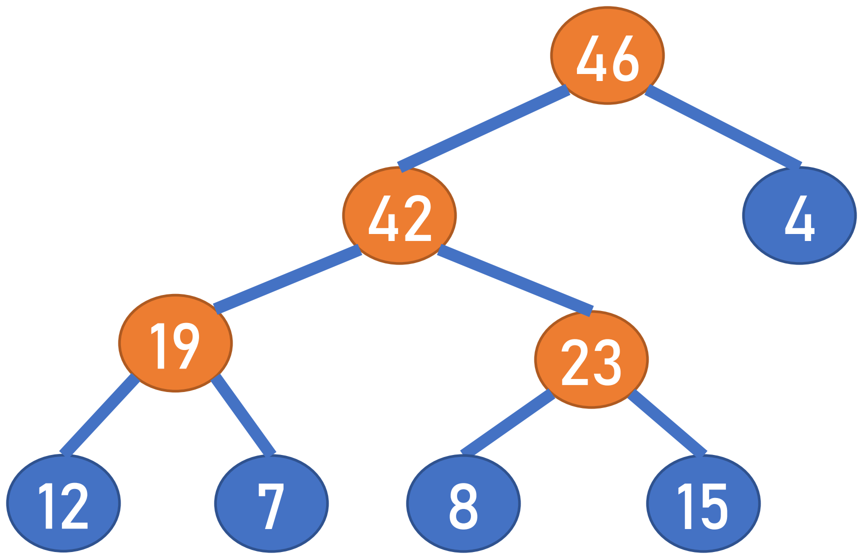
- **We have pebble piles and want to merge them into one pile, but**
  - We can only merge two of them at a time
  - Merging two piles of size a and b cost you a+b units of energy
- **Use a tree to represent the trace of merging**



Energy cost: 19 +23 +42

Merge pebbles – another solution

- **We have pebble piles and want to merge them into one pile, but**
  - We can only merge two of them at a time
  - Merging two piles of size a and b cost you a+b units of energy
- **Use a tree to represent the trace of merging**



Energy cost: 19 +23 +42 +46 =130

Merge pebbles – Can you design a greedy solution?

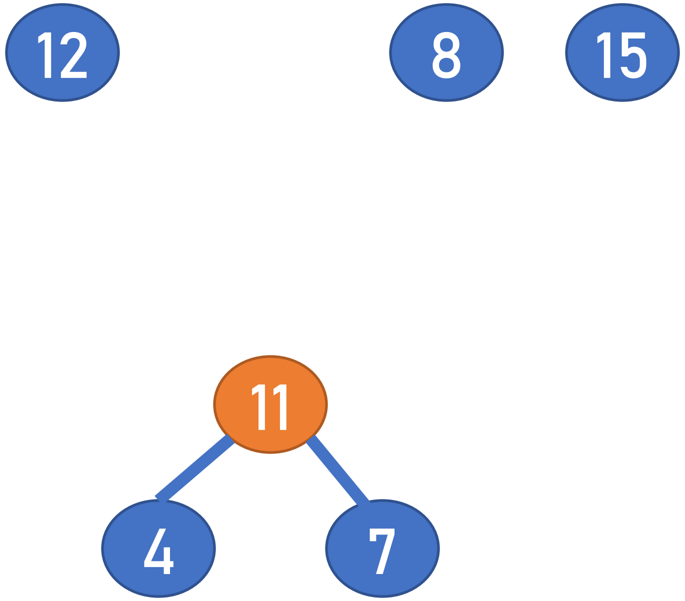
- **We have pebble piles and want to merge them into one pile, but**
  - We can only merge two of them at a time
  - Merging two piles of size a and b cost you a+b units of energy
- **Use a tree to represent the trace of merging**

Merge pebbles – greedy solution

- **We have pebble piles and want to merge them into one pile, but**
  - We can only merge two of them at a time
  - Merging two piles of size a and b cost you a+b units of energy
- **Always merge the two with the fewest pebbles!**

Merge pebbles –greedy solution?

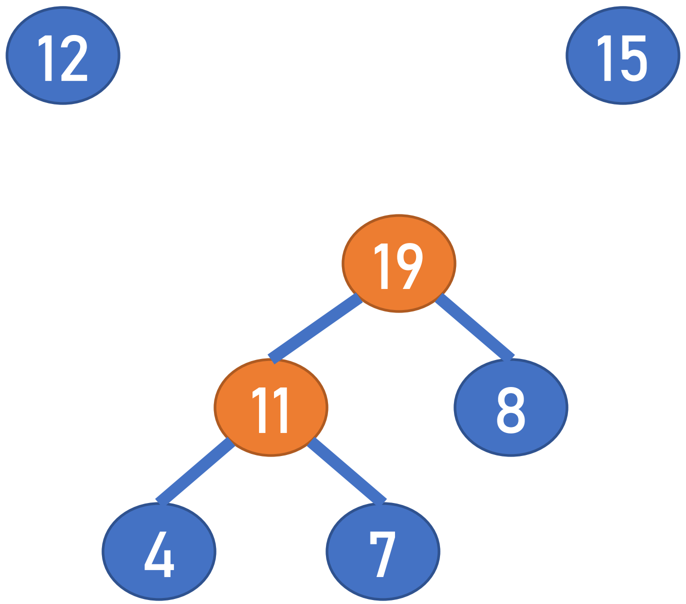
- **We have pebble piles and want to merge them into one pile, but**
  - We can only merge two of them at a time
  - Merging two piles of size a and b cost you a+b units of energy
- **Always merge the two with the fewest pebbles!**



Energy cost: 11

Merge pebbles –greedy solution?

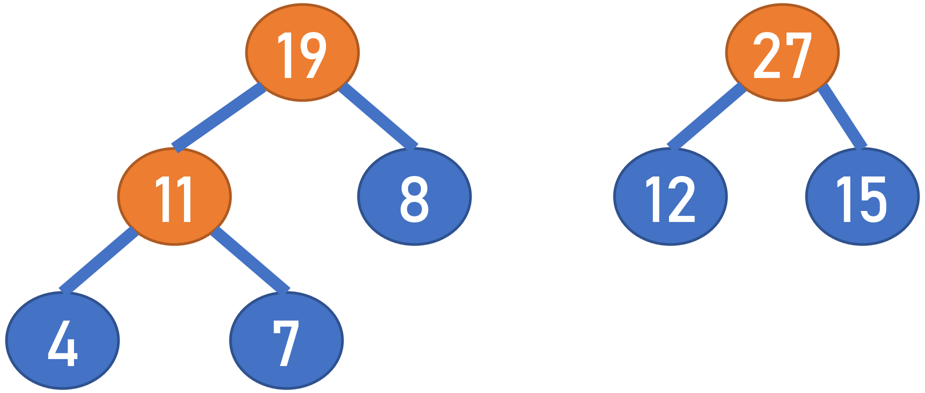
- **We have pebble piles and want to merge them into one pile, but**
  - We can only merge two of them at a time
  - Merging two piles of size a and b cost you a+b units of energy
- **Always merge the two with the fewest pebbles!**



Energy cost: 11 +19

Merge pebbles –greedy solution?

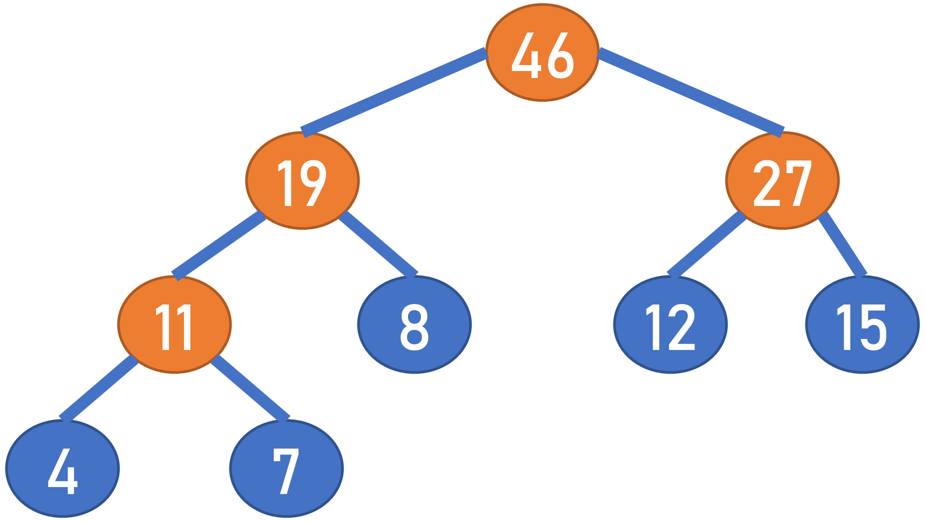
- **We have pebble piles and want to merge them into one pile, but**
  - We can only merge two of them at a time
  - Merging two piles of size a and b cost you a+b units of energy
- **Always merge the two with the fewest pebbles!**



Energy cost: 11 +19 +27

Merge pebbles –greedy solution?

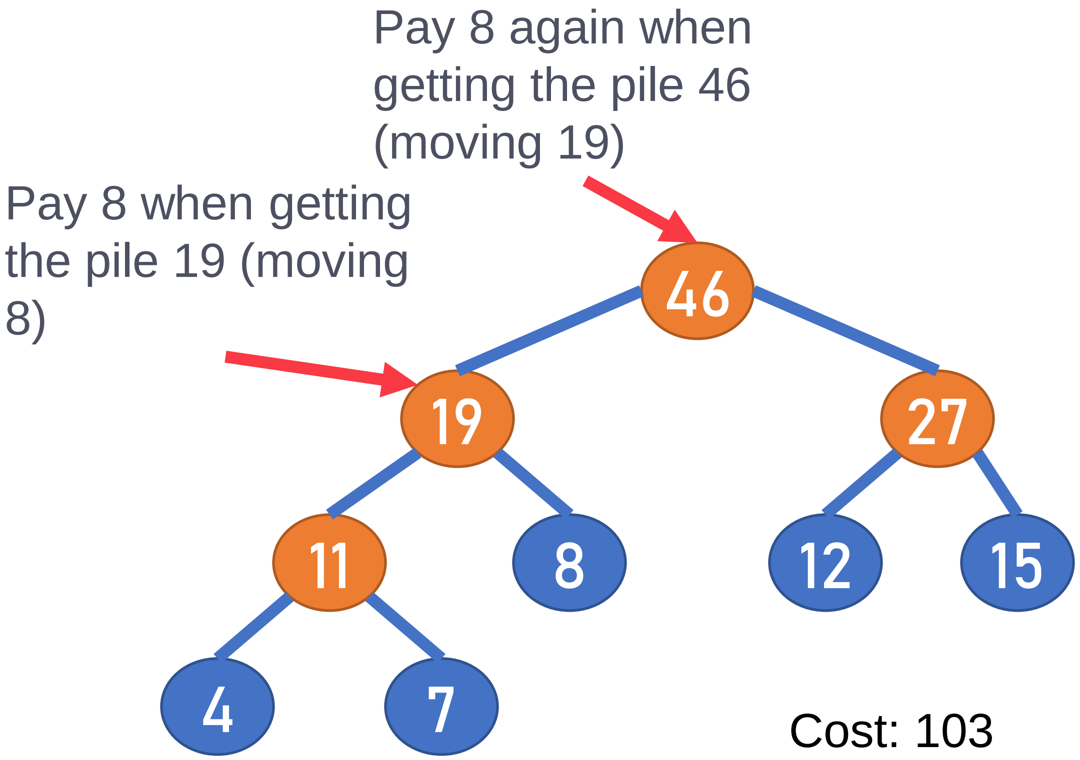
- **We have pebble piles and want to merge them into one pile, but**
  - We can only merge two of them at a time
  - Merging two piles of size a and b cost you a+b units of energy
- **Always merge the two with the fewest pebbles!**

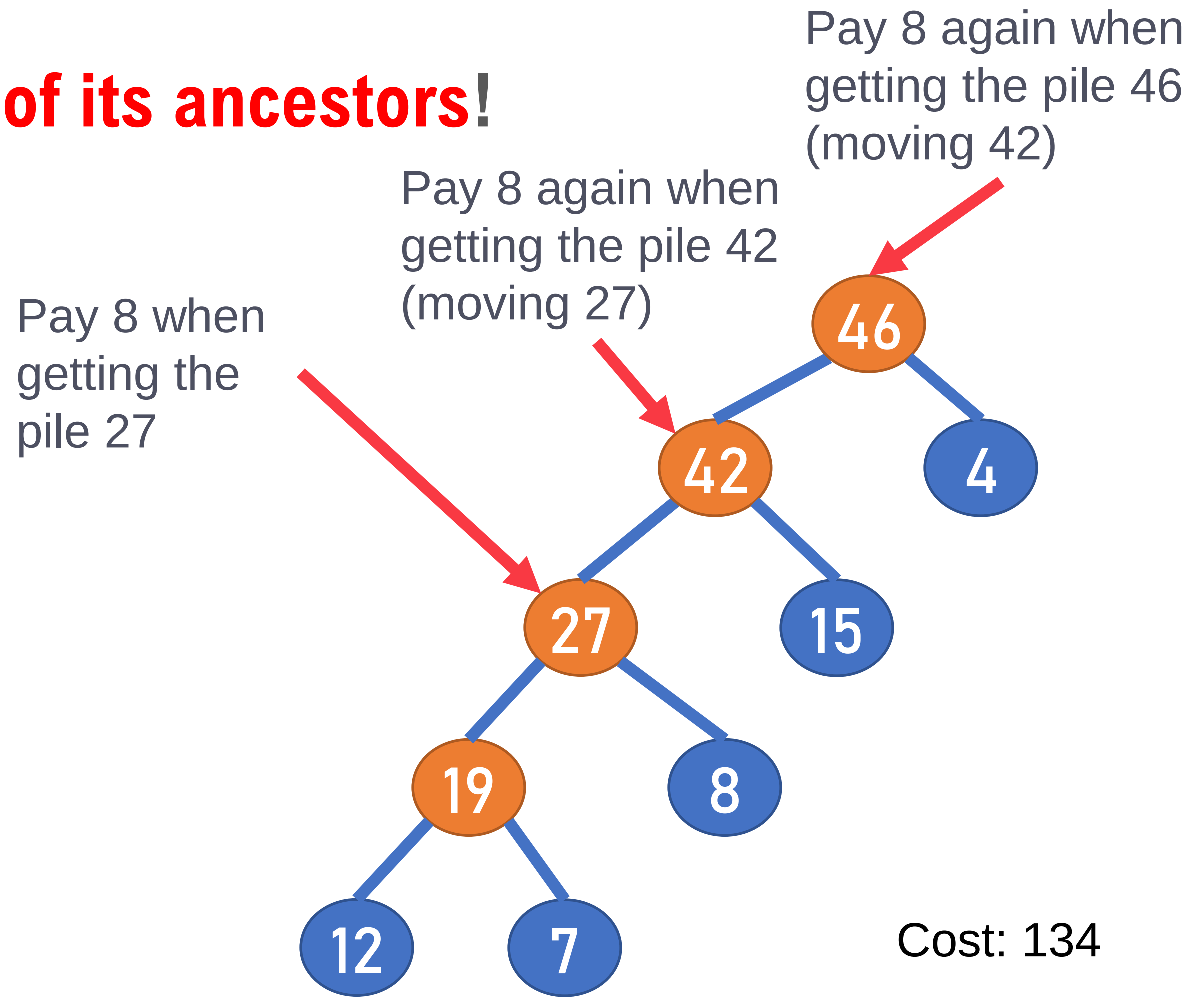


Energy cost: 11 +19 +27 +46 =103

Merge pebbles – Why greedy is good?

- You may need to move a pile **multiple times** (its size counts in the cost for multiple times)
- The pile size will be charged at **all of its ancestors**!

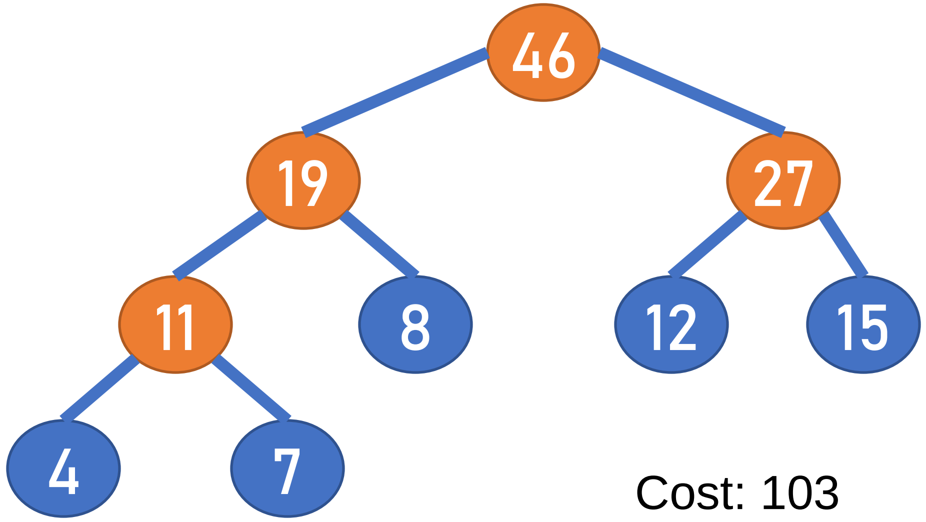




Merge pebbles – Why greedy is good?

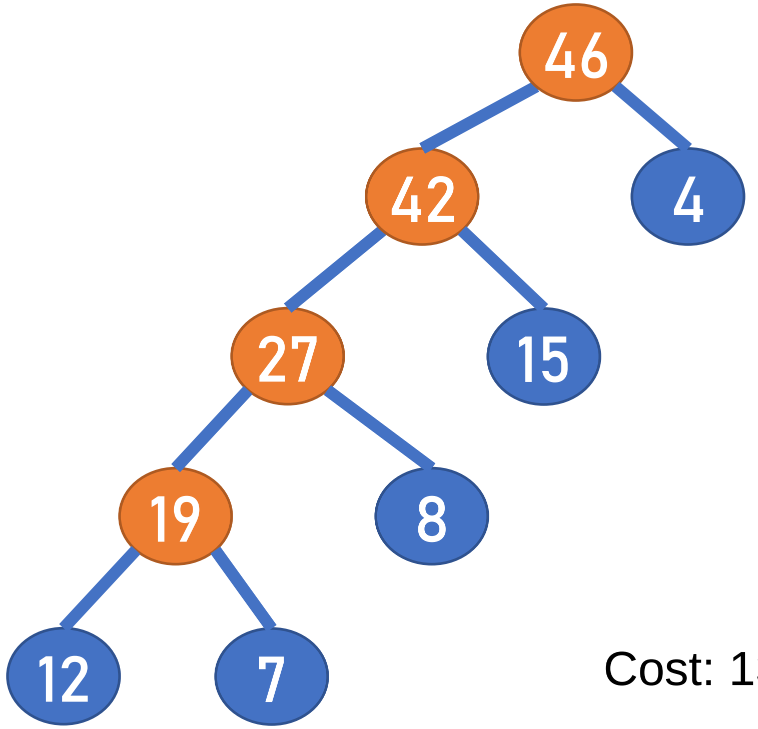
- **You may need to move a pile multiple times (its size counts in the cost for multiple times)**
- **The pile size will be charged at all of its ancestors!**
- **How many times do you need to move the pile 8?**
  - The depths of it! (the number of ancestors)

Total cost: $4 \times 3 + 7 \times 3 + 8 \times 2 + 12 \times 2 + 15 \times 2 = 103$



Cost: 103

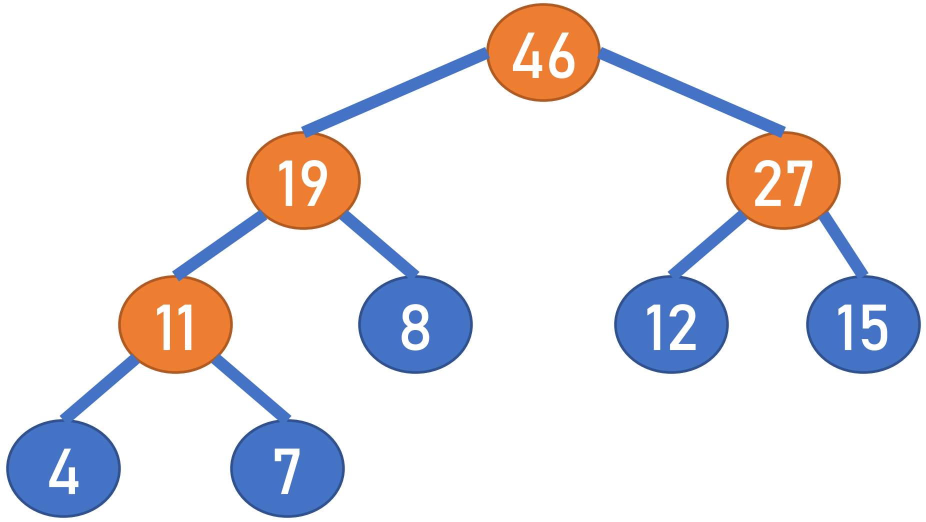
Total cost: $4 \times 1 + 7 \times 4 + 8 \times 3 + 12 \times 4 + 15 \times 2 = 134$



Cost: 134

Merge pebbles – Why greedy is good?

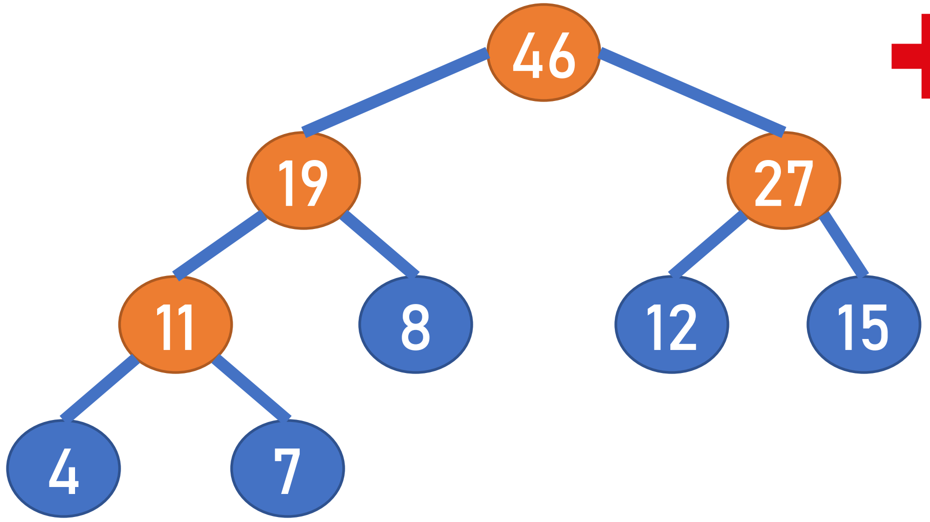
- **You may need to move a pile multiple times (its size counts in the cost for multiple times)**
- **The pile size will be charged at all of its ancestors!**
- **How many times do you need to move the pile 8?**
  - The depths of it! (the number of ancestors)
- $cost = \sum_{leaf\ t \in T} t \times d(t)$ $\quad d(t)$ is the depth of pile t in the merging tree



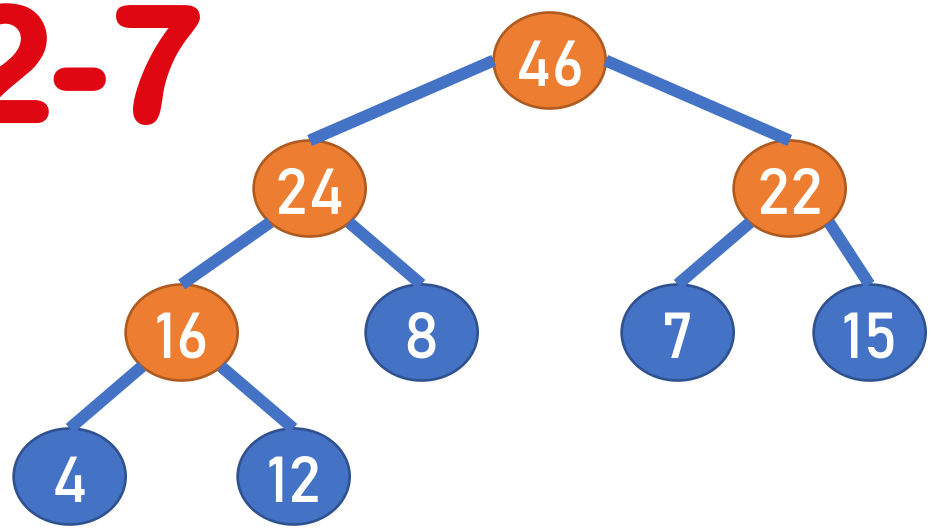
Total cost: $4 \times 3 + 7 \times 3 + 8 \times 2 + 12 \times 2 + 15 \times 2 = 103$

Merge pebbles – Why greedy is good?

- $cost = \sum_{leaf\ t \in T} t \times d(t)$ $d(t)$ is the depth of pile t in the merging tree



+12-7

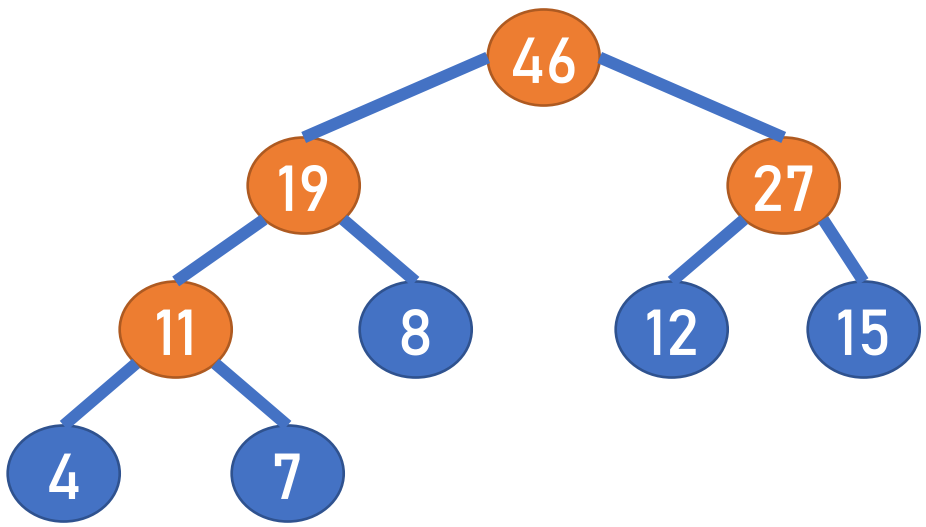


Total cost: $4 \times 3 + 7 \times 3 + 8 \times 2 + 12 \times 2 + 15 \times 2 = 103$

Total cost: $4 \times 3 + 7 \times 2 + 8 \times 2 + 12 \times 3 + 15 \times 2 = 108$

Merge pebbles – Why greedy is good? (intuitively)

- $cost = \sum_{leaf\ t \in T} t \times d(t)$ $\quad d(t)$ is the depth of pile t in the merging tree



Total cost: $4 \times 3 + 7 \times 3 + 8 \times 2 + 12 \times 2 + 15 \times 2 = 103$

- **Should make two smallest piles deepest**
- **There are always two leaves in the deepest level**
- **We can always merge them first**

- **Optimal substructure: the problem size decreases by 1**
  - **n-1 piles of pebbles to merge, minimize energy**

- **A formal prove is in the textbook, and more explanation will be given later in this lecture**

However, why do we care about moving pebble piles???

Huffman Codes

Encoding

- **How data is represented?**
- **Fixed-size codes, e.g., ASCII**
  - A: 1000001 (65)
  - B: 1000010 (66)
- **Variable-size codes, e.g., Morse Codes**
  - A: ●▬
  - B: ▬●●●
  - E: ●
  - T: ▬

Example: Morse Code

"SOS":

● ● ● ▬ ▬ ▬ ● ● ●

A ●▬
B ▬●●●
C ▬●▬●
D ▬●●
E ●
F ●●▬●
G ▬▬●
H ●●●●
I ●●
J ●▬▬▬
K ▬●▬
L ●▬●●
M ▬▬
N ▬●
O ▬▬▬
P ●▬▬●
Q ▬▬●▬
R ●▬●
S ●●●
T ▬
U ●●▬
V ●●●▬
W ●▬▬
X ▬●●▬
Y ▬●▬▬
Z ▬▬●●

IJS: (..)(.---)(...)

STZE: (…)(-)(--..)(.)

IAGI: (..)(.-)(--.)(..)

VMS: (...-)(--)(...)

Prefix Codes

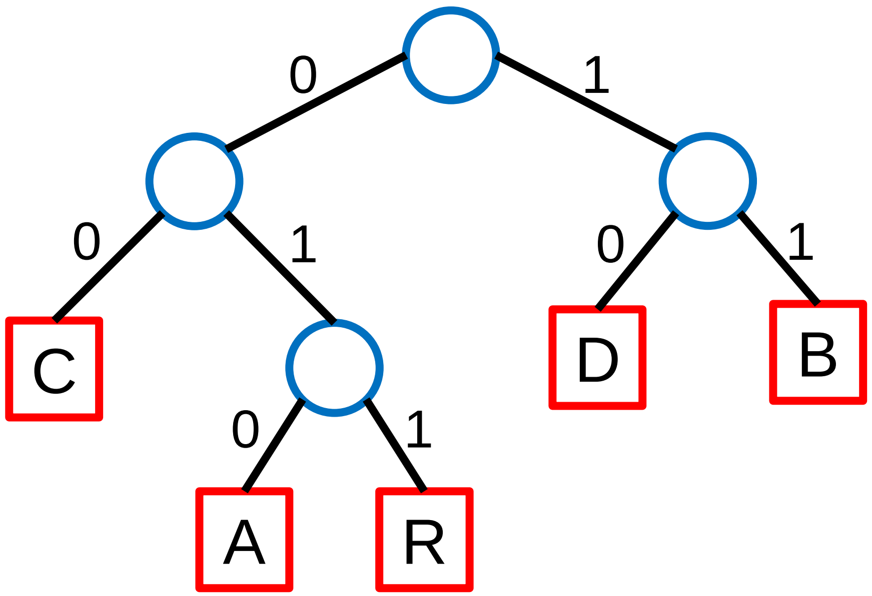
- **No code is allowed to be a prefix of another code**
- **To encode, simply concatenate all the codes**
- **Decoding does not entail any ambiguity**
- **Example:**
  - Message 'JAVA'
  - a = "0", j = "11", v = "10"
  - Encoded message "110100"
  - Decoding "110100" – greedily decode it!

character	Prefix code	Non prefix code
A	00	00
B	01	001
C	101	11
D	100	111
E	11	01

0011101

Trie

- We can use a trie to find prefix codes
- the characters are stored at the external nodes
- a left child (edge) means 0
- a right child (edge) means 1
- No code can be prefix of another code



A=010

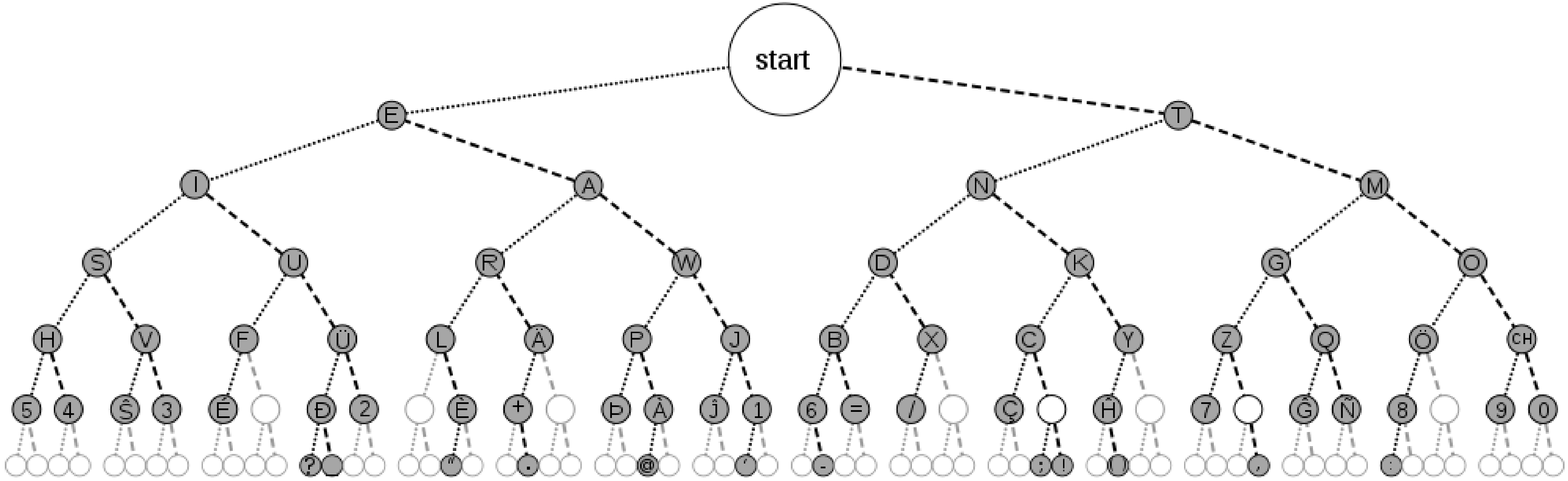
B=11

C=00

D=10

R=011

Morse code (not a prefix code)



Source: Wikipedia

Example of Decoding

- Encoded text: 00010011
- Text = ?
- Encoded text: 0001101011
- Text = ?

A=010

B=11

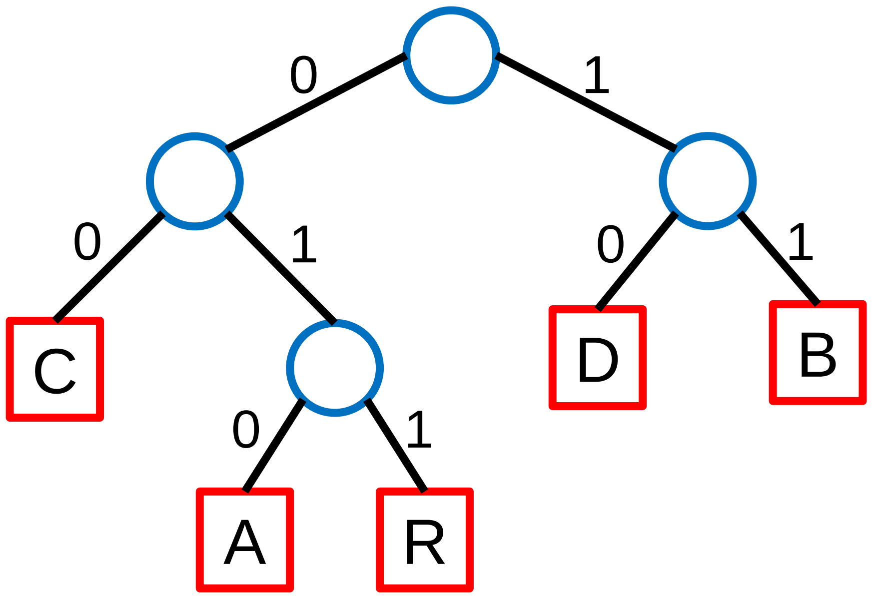
C=00

D=10

R=011

Example of Decoding

- encoded text: 01011011010000101001011011010
- Very expensive to check all possibilities
- Use the tree!
- text: ABRACADABRA (11 characters)
  - ASCII: 77 bits
  - Our encoding: 29 bits



A=010

B=11

C=00

D=10

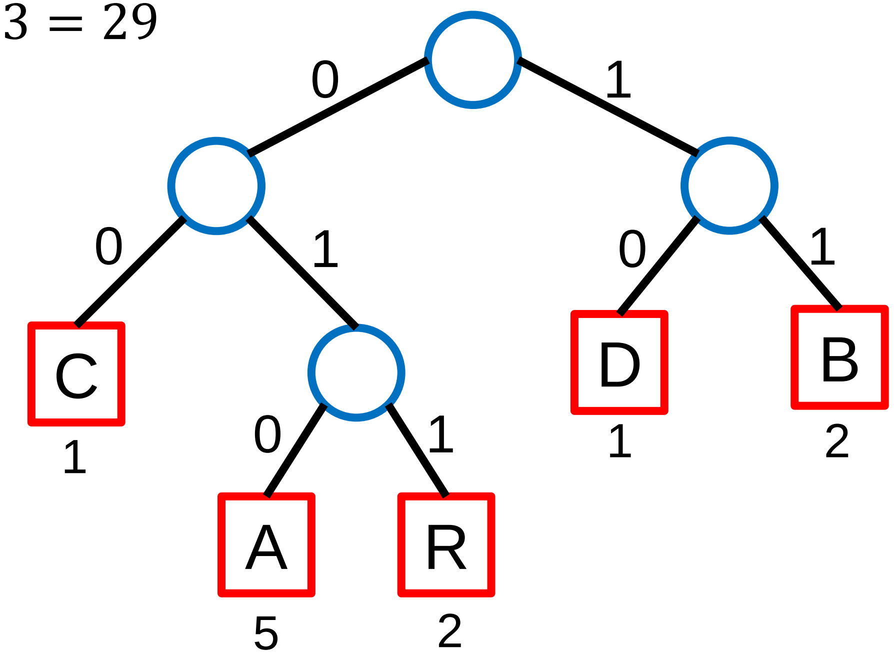
R=011

Example of Encoding

- Message: 'ABRACADABRA' (11 characters)
- Encoded message: '01011011010000101001011011010'
- Length: 29 bits

Total length: $5 \times 3 + 2 \times 2 + 1 \times 2 + 1 \times 2 + 2 \times 3 = 29$

The length of the code for character c is just its depth $d(c)$!



A=010

B=11

C=00

D=10

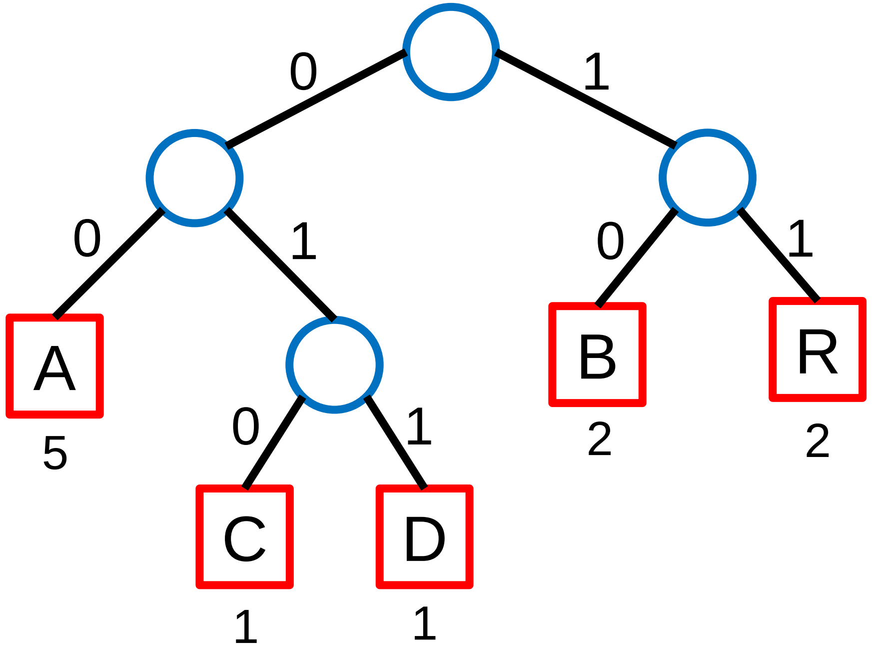
R=011

Example of Encoding

- **Message: 'ABRACADABRA' (11 characters)**
- **Encoded message: '001011000100001100101100'**
- **Length: 24 bits**

Total length: $5 \times 2 + 1 \times 3 + 1 \times 3 + 2 \times 2 + 2 \times 2 = 24$

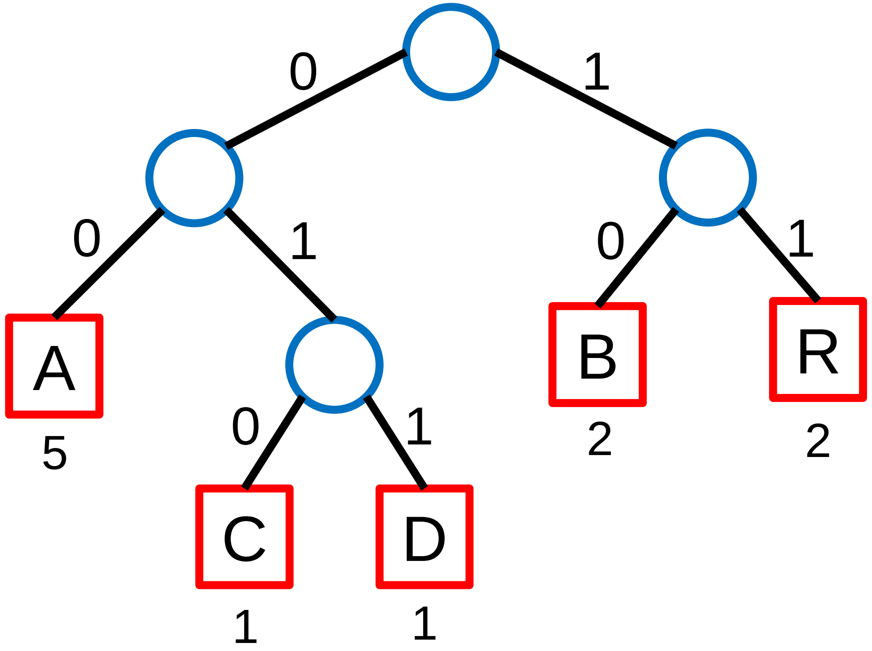
The length of the code for character c is just its depth $d(c)$!



Optimal Encoding Problem

- Given a set C of n characters, for each character $c \in C$. Let $c.freq$ be the frequency of c in the file
- We would like to find a prefix encoding for each $c \in C$ with a length $d(c)$ such that we minimize the total cost

$$\boldsymbol{cost} = \sum_{c \in C} c.freq \times \boldsymbol{d}(c)$$

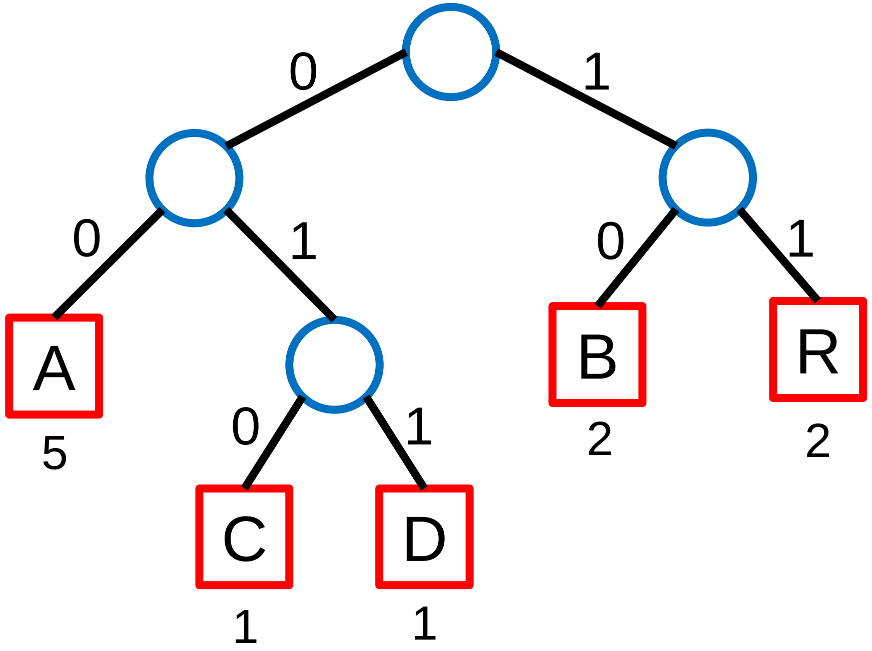


Total length: $5 \times 2 + 1 \times 3 + 1 \times 3 + 2 \times 2 + 2 \times 2 = 24$

Optimal Encoding Problem

- Given a set C of n characters, for each character $c \in C$. Let $c.freq$ be the frequency of c in the file.
- We would like to find a prefix encoding for each $c \in C$ with a length $d(c)$ such that we minimize the total cost

$$\boldsymbol{cost} = \sum_{c \in C} c.freq \times \boldsymbol{d}(c)$$



- That's the same with our pebble merging problem!
    - Frequency = initial pebble pile size
- Solution: Huffman Codes

Huffman codes

- **Find the two characters with the least frequency x and y**
  - Find to piles of pebbles with smallest size
- **Combine them in to one temporary character (internal node) with frequency $x + y$**
  - Combine them into one pile of size $x + y$
- **Repeat until there is only one node**
  - Repeat until there is only one pile

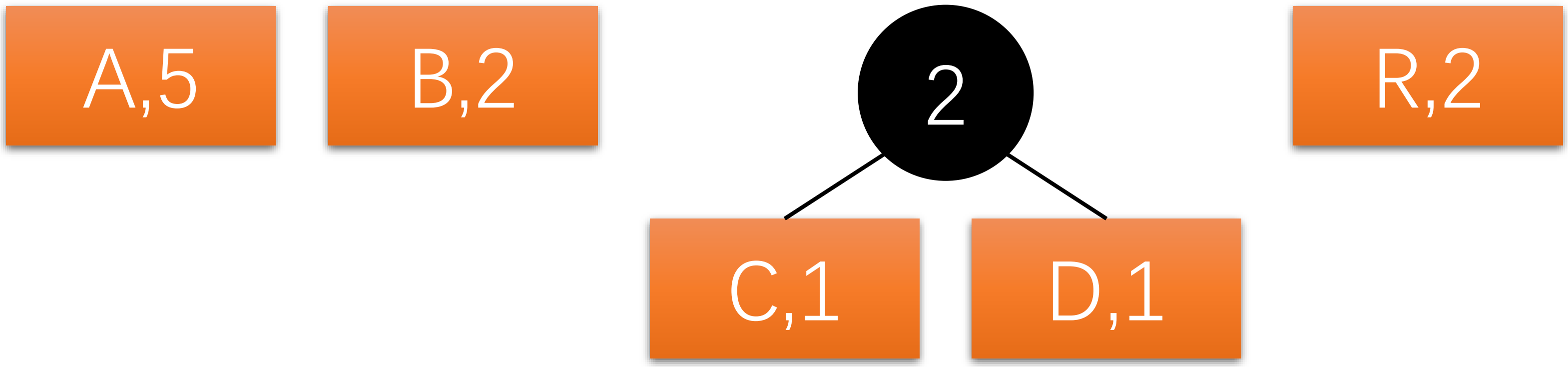
Example

"ABRACADABRA"

A,5	B,2	C,1	D,1	R,2

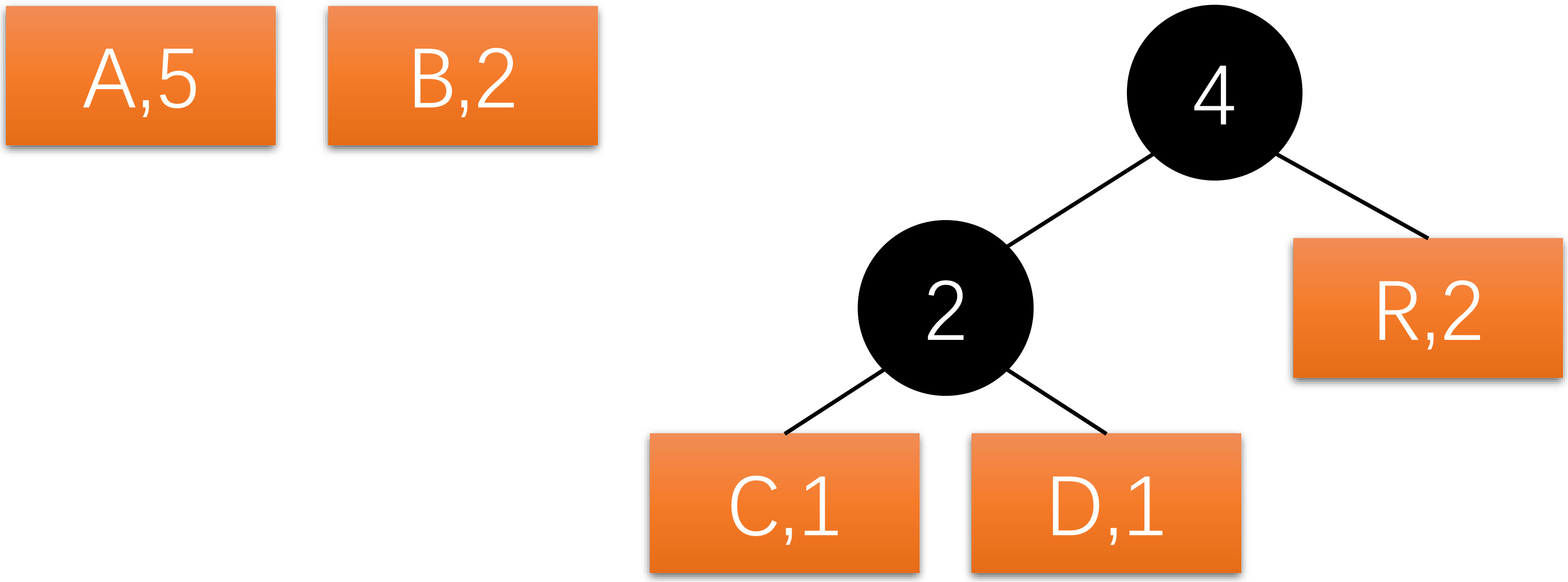
Example

"ABRACADABRA"



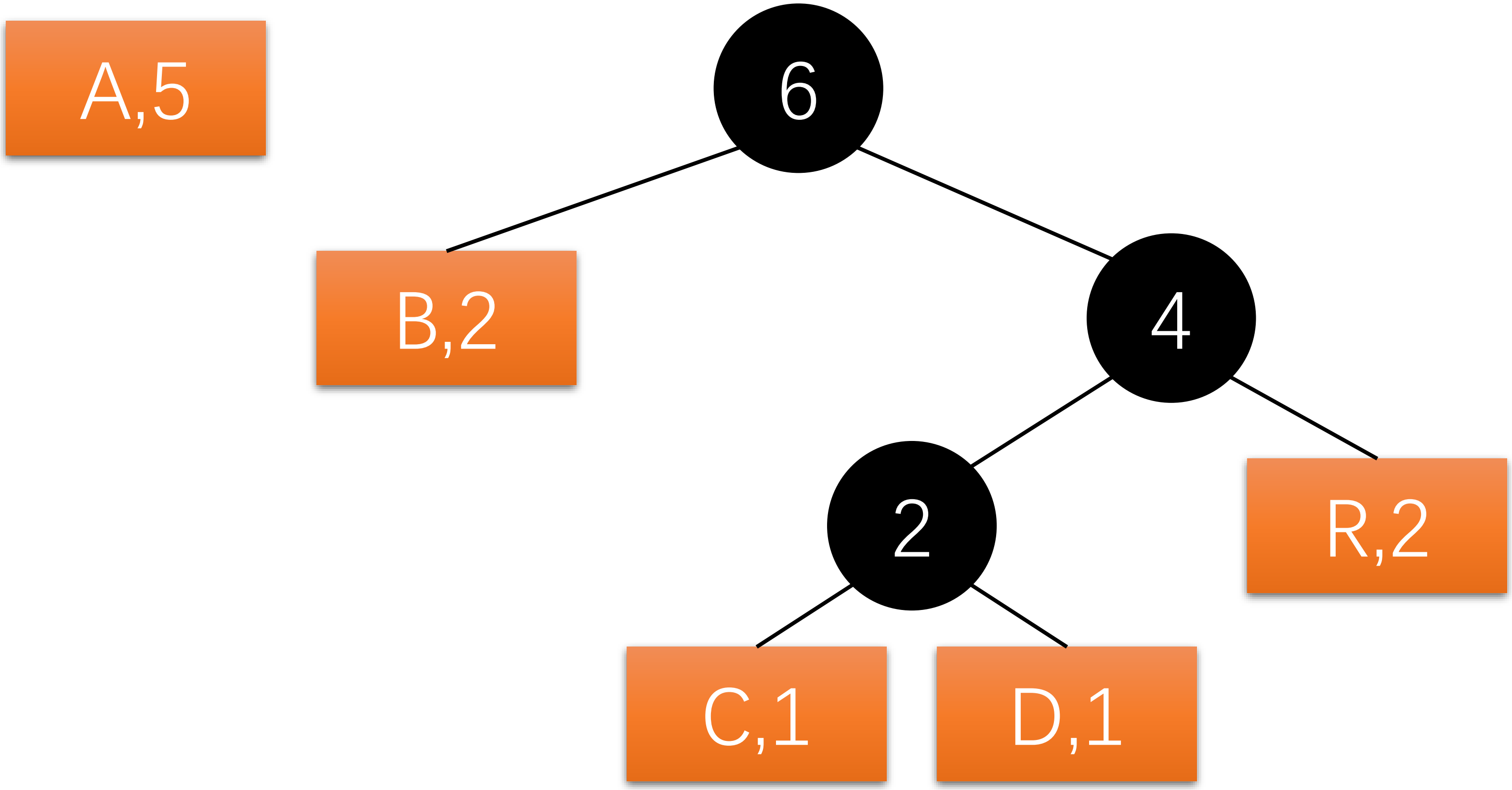
Example

"ABRACADABRA"

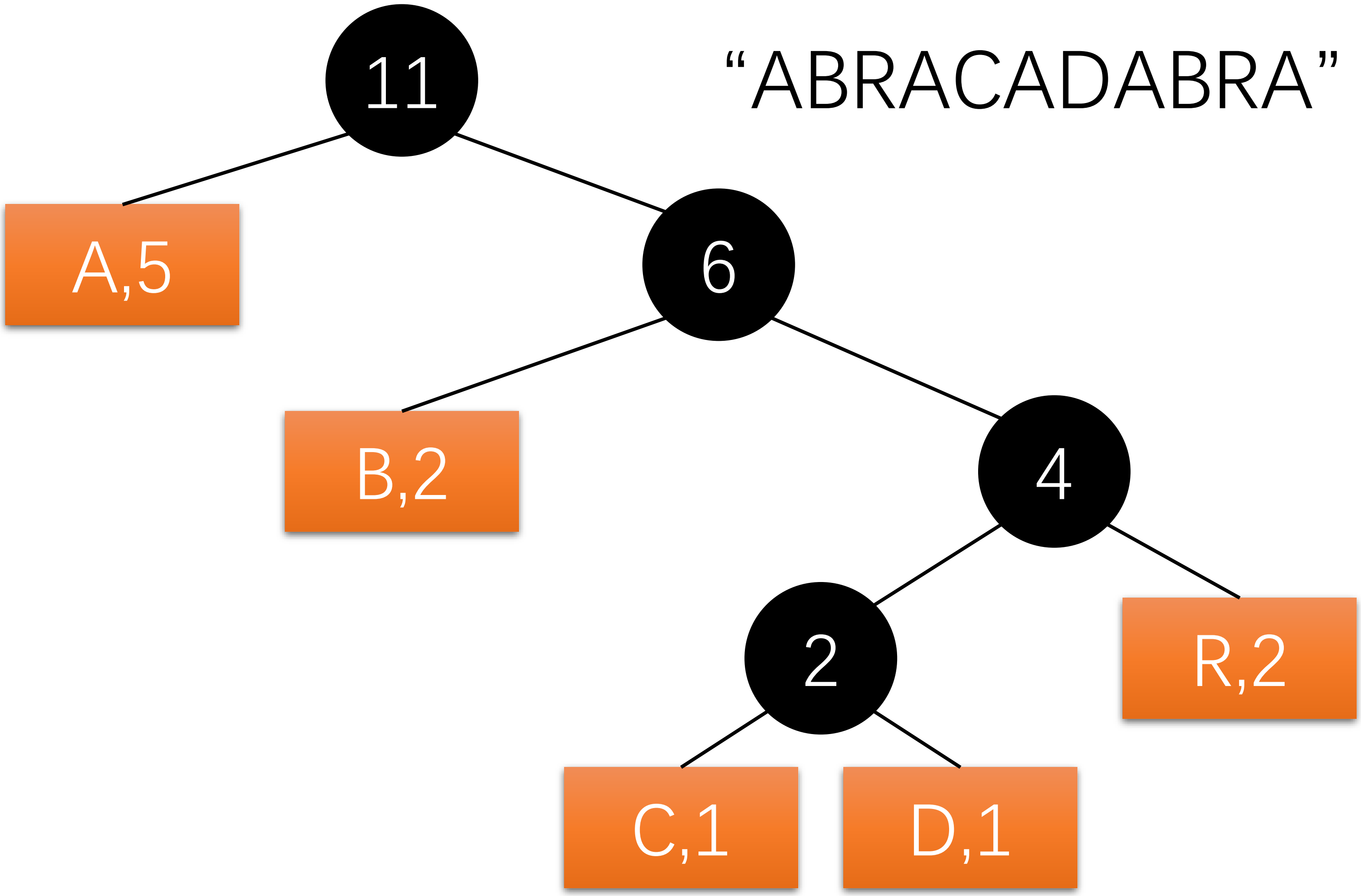


Example

"ABRACADABRA"

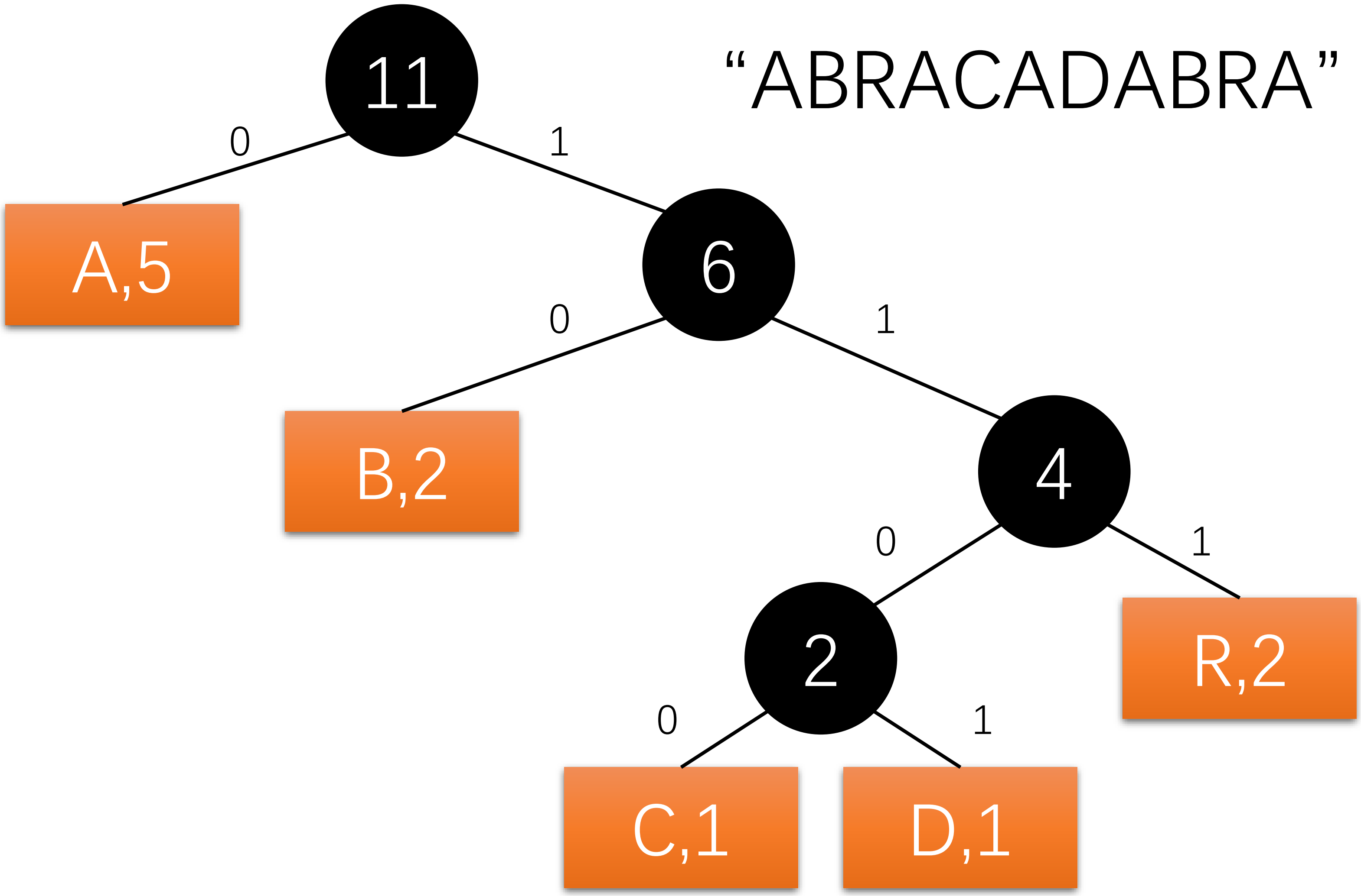


Example



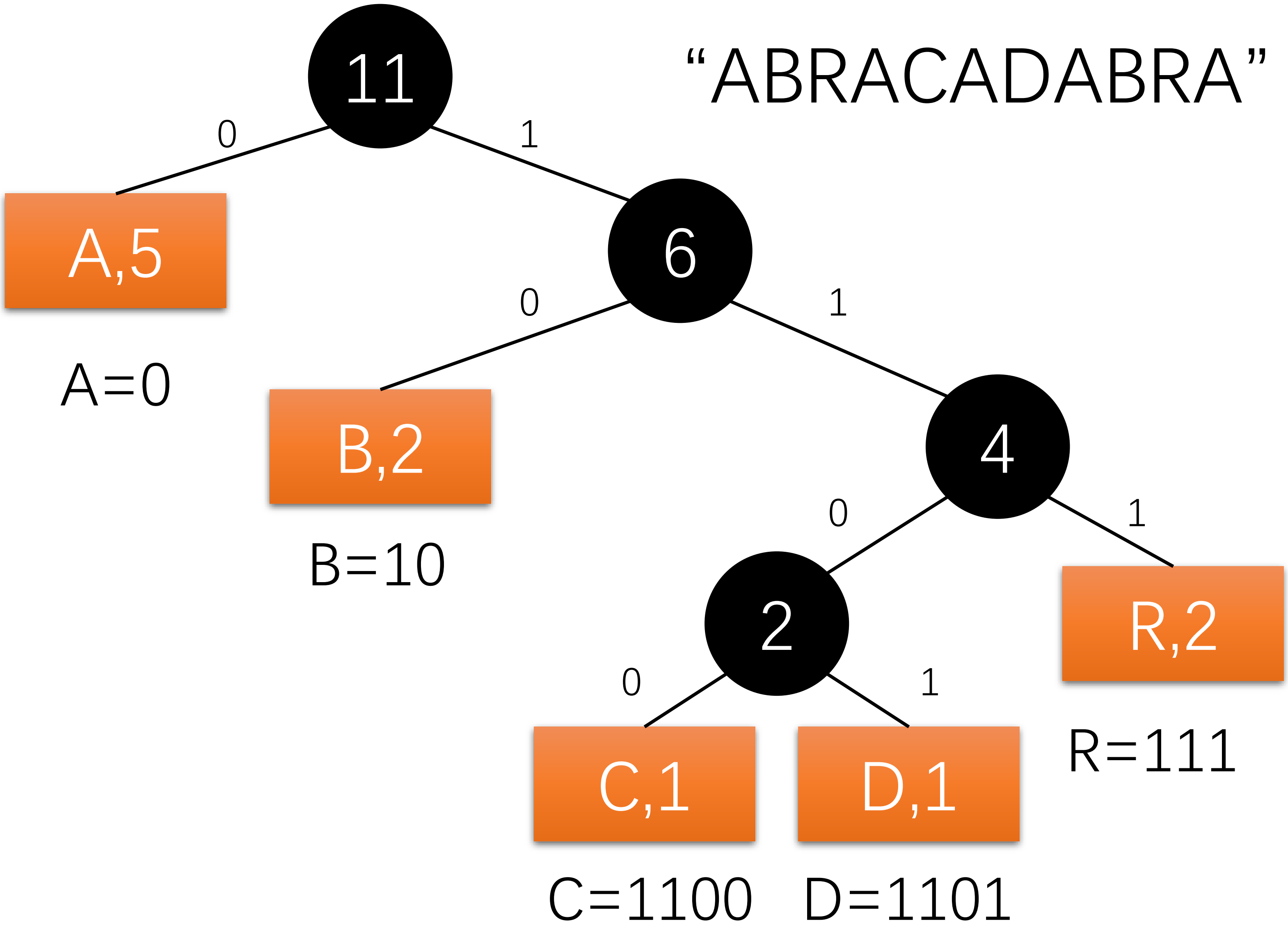
“ABRACADABRA”

Example



"ABRACADABRA"

Example



"ABRACADABRA"
11
0
1
A,5
A=0
6
0
1
B,2
B=10
4
0
1
2
R,2
R=111
0
1
C,1
D,1
C=1100
D=1101

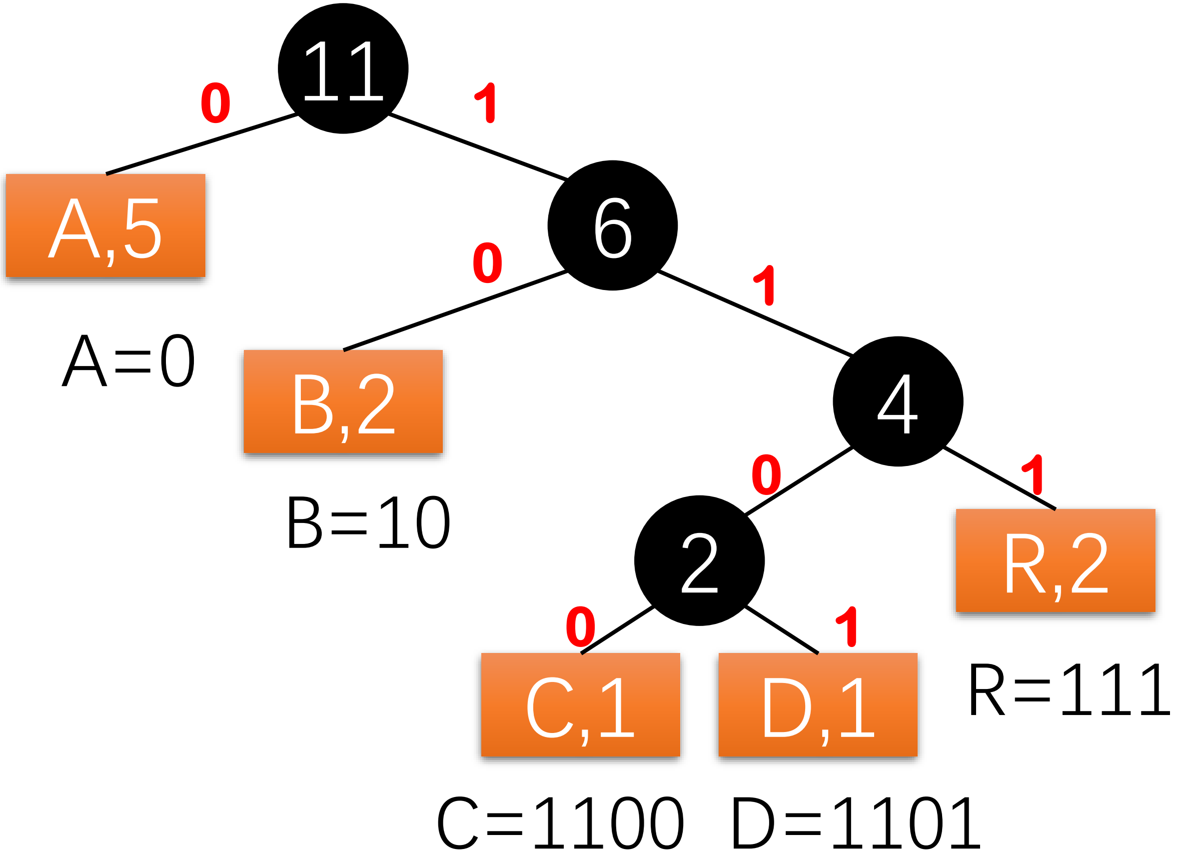
Encoding

"ABRACADABRA"

0 10 111 0 1100 0 1101 0 10 111 0

Length= 23

Optimal!



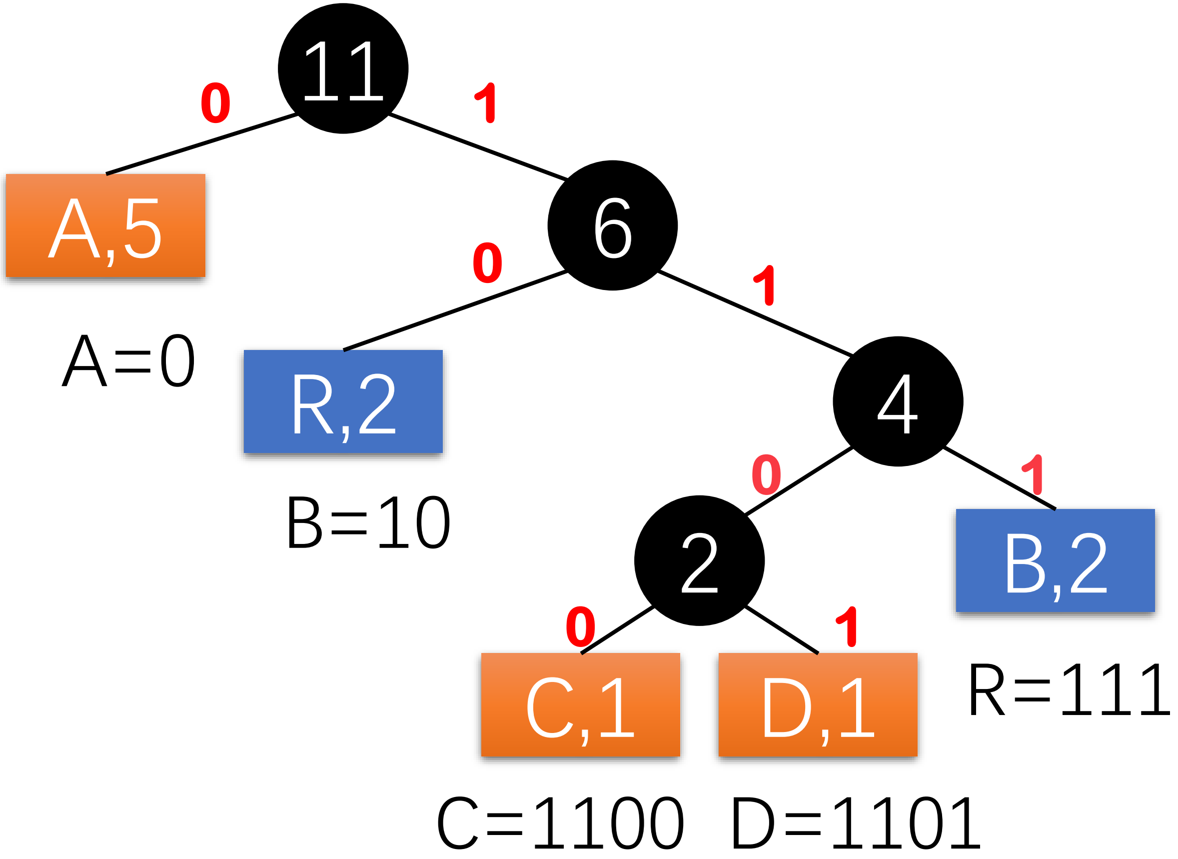
Encoding

"ABRACADABRA"

0 111 10 0 1100 0 1101 0 111 10 0

Length= 23

Optimal!



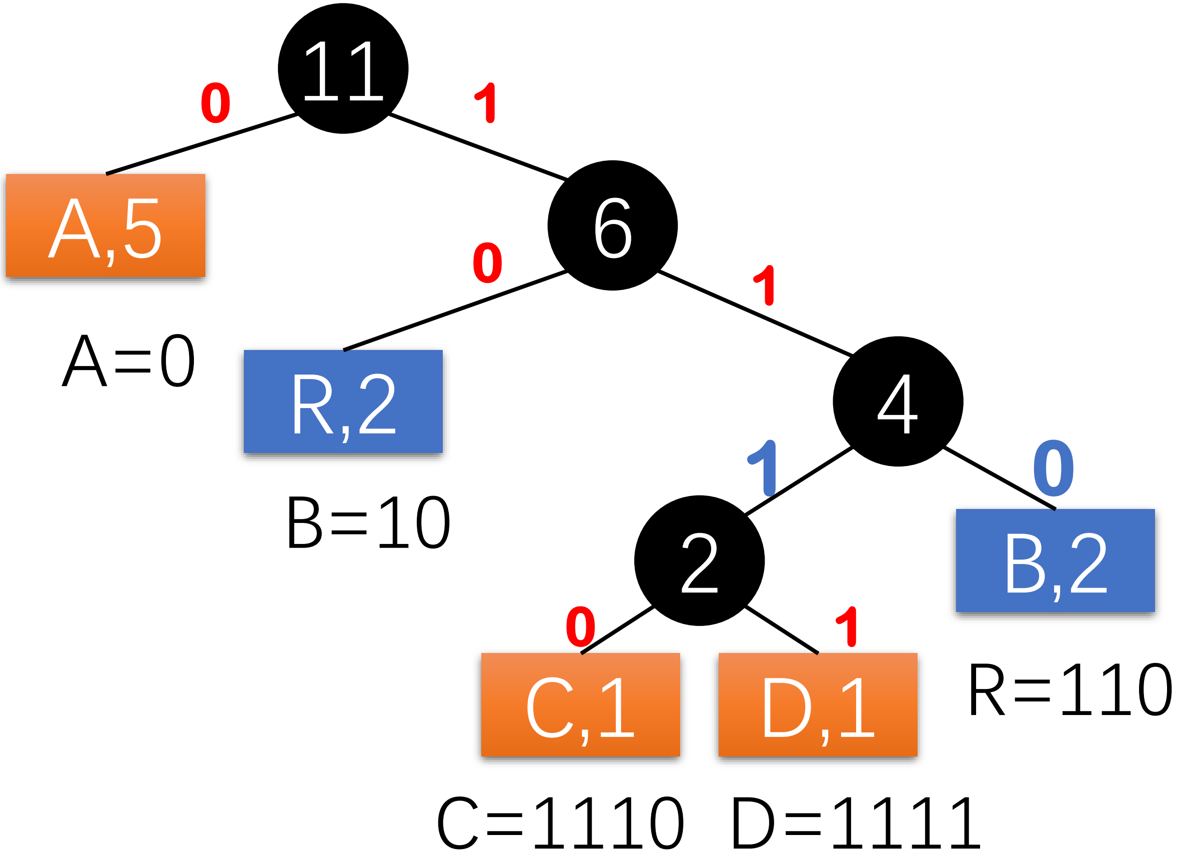
Encoding

"ABRACADABRA"

0 110 10 0 1110 0 1111 0 110 10 0

Length= 23

Optimal!



Construction of Huffman Tree

Note: Can also be done in linear time

- **Huffman(C)**
  - n=|C|
  - Q=C // construct a priority queue of all character's frequency → $\Theta(n \log n)$
  - for i = 1 to n-1
    - allocate a new node z
    - z.left = x = Extract-Min(Q)
    - z.right = y = Extract-Min(Q)
    - z.freq = x.freq + y.freq
    - Insert(Q, z)
  - return Extract-Min(Q) // Root of the tree

n

$\Theta(\log n)$

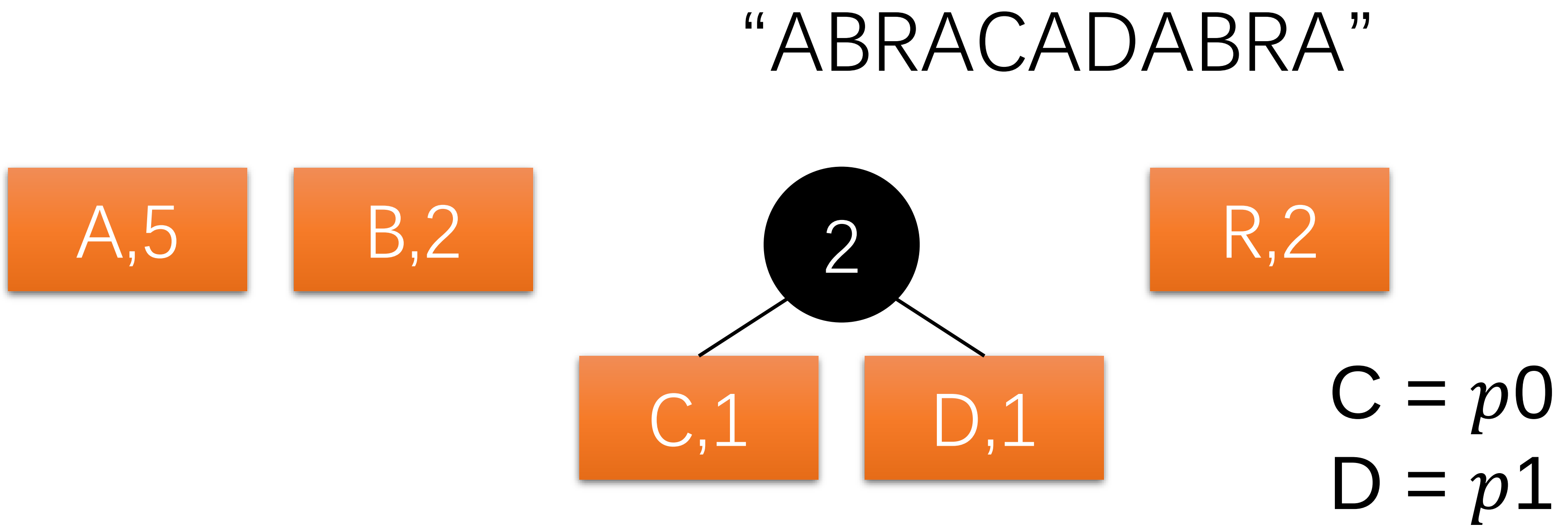
$$T(n) = \Theta(n \log n)$$

Optimality of Huffman Codes

- **Greedy-choice**
  - The greedy choice yields an optimal solution.
- **Optimal substructure**
  - The optimal solution for the bigger problem contains the optimal solution of the sub-problem.

- **Similar to the pebble merging**
- **Detailed proof in the textbook**

Optimal substructure



Merging C and D

- They must share the same prefix p, and ending with 0 and 1, respectively
- Consider them as a whole: the frequency of p is 1+1=2.
- Create a new node (represents the prefix p) of frequency 2

Optimal substructure

"ABRA(p0)A(p1)ABRA"

A,5 | B,2 | p, 2 | R,2

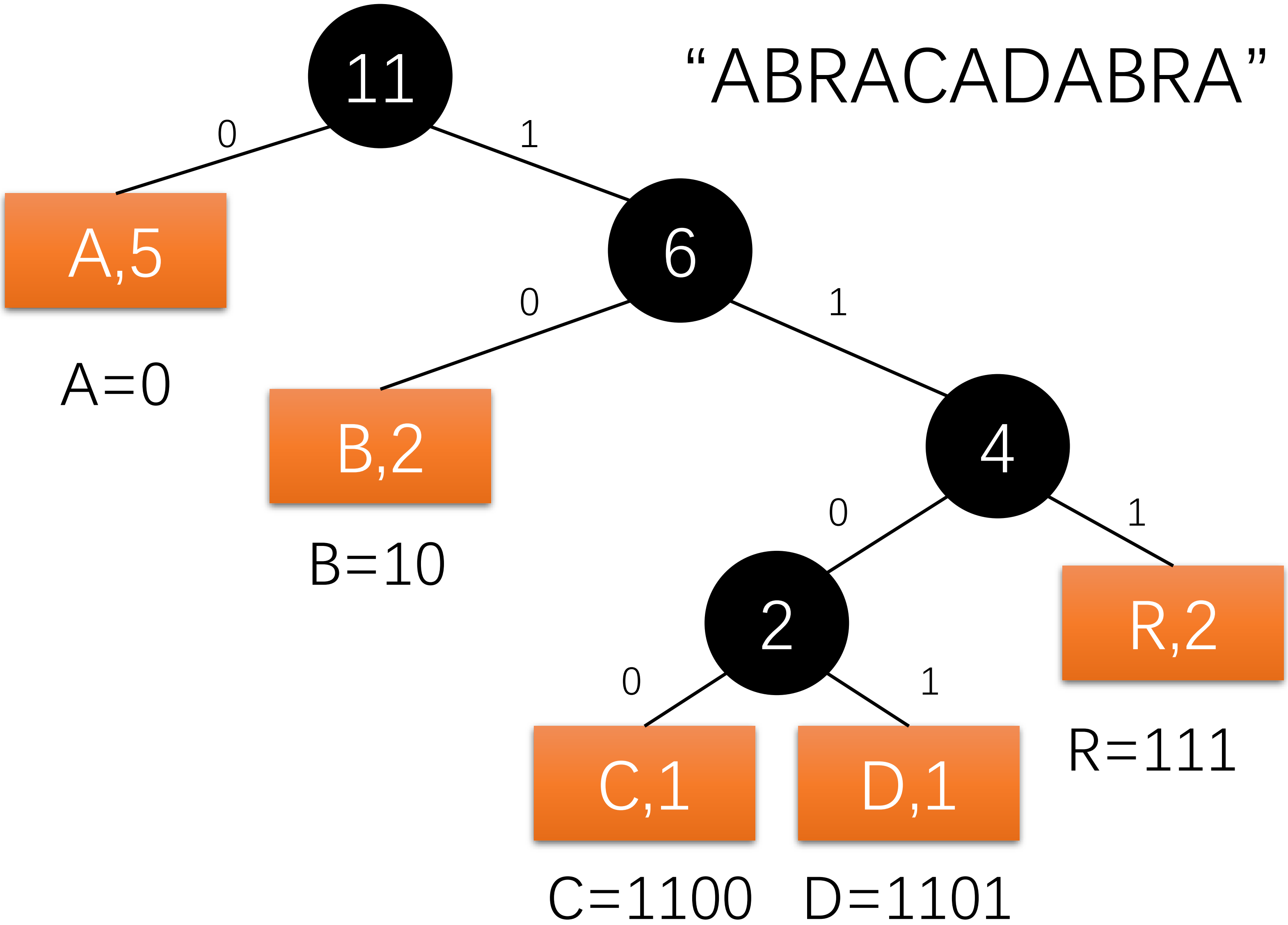
C = $p0$

D = $p1$

Merging C and D

- They must share the same prefix p, and ending with 0 and 1, respectively
- Consider them as a whole: the frequency of p is 1+1=2.
- Create a new node (represents the prefix p) of frequency 2
- Repeat the process – find the string for p recursively

Example



"ABRACADABRA"

Recall:
C = p0
D = p1
p is 110

What cannot be solved by greedy strategies?

- Different candies have different "values" (say, how much you like them)
- With a fixed budget of S dollars, how to maximize the total value?
- No known greedy algorithm can solve this ☹
- A lot of variants: 0/1 knapsack, unlimited knapsack, k-knapsack, …

$5
Value = 2

Value = 1
$2

Value = 2

$4

Value = 4

$7

Value = 20
$15

$5 Value = 3

$1 Value = 1

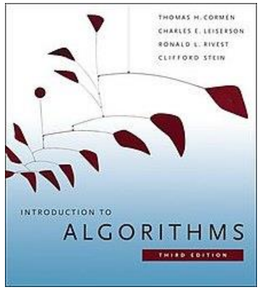
$7
Value = 5

$9
Value = 3

Knapsack problem

- Your little brother is attending university this year
- Unfortunately, he did not get an offer from UCR, and he has to go to the east coast, and needs to take a flight

$50, 1lb



$70, 5lb

$1500, 8lb

$80, 2lb

A simplified case: unlimited knapsack

- Overall weight limit: 8 lb, we can take an unlimited number of each item
- Item 1: 5 lb, $150
- Item 2: 4 lb, $100
- Item 3: 2 lb, $10

- Solution 1: Item 1 + Item 3, value: $160
- Solution 2: Item 2 * 2, value: $200

- Greedy strategy does not provide the optimal solution
- A naïve solution? Try all possibilities!

A naïve algorithm

Item 1: 5 lb, $150
Item 2: 4 lb, $100
Item 3: 2 lb, $10

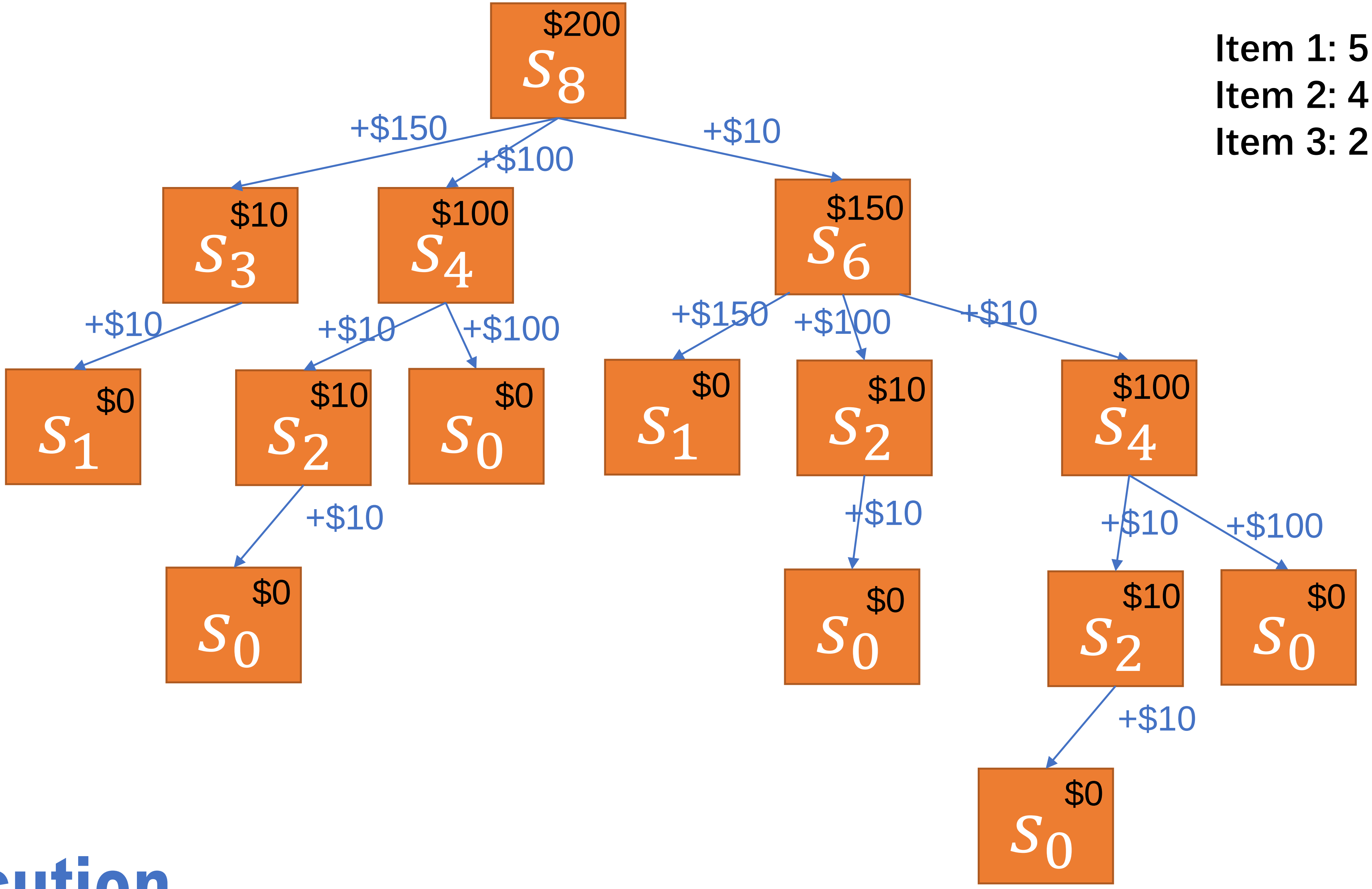
suitcase(8):
Case 1: first put item 1,
total value = suitcase(3) + 150
Case 2: first put item 2,
total value = suitcase(4) + 100
Case 3: first put item 3,
total value = suitcase(6) + 10

```
int suitcase(int leftWeight) {
    int curBest = 0;
    foreach item of (weight, value)
        if (leftWeight >= weight)
            curBest = max(curBest, suitcase(leftWeight - weight) + value);
    return curBest;
}
```

Recursive call

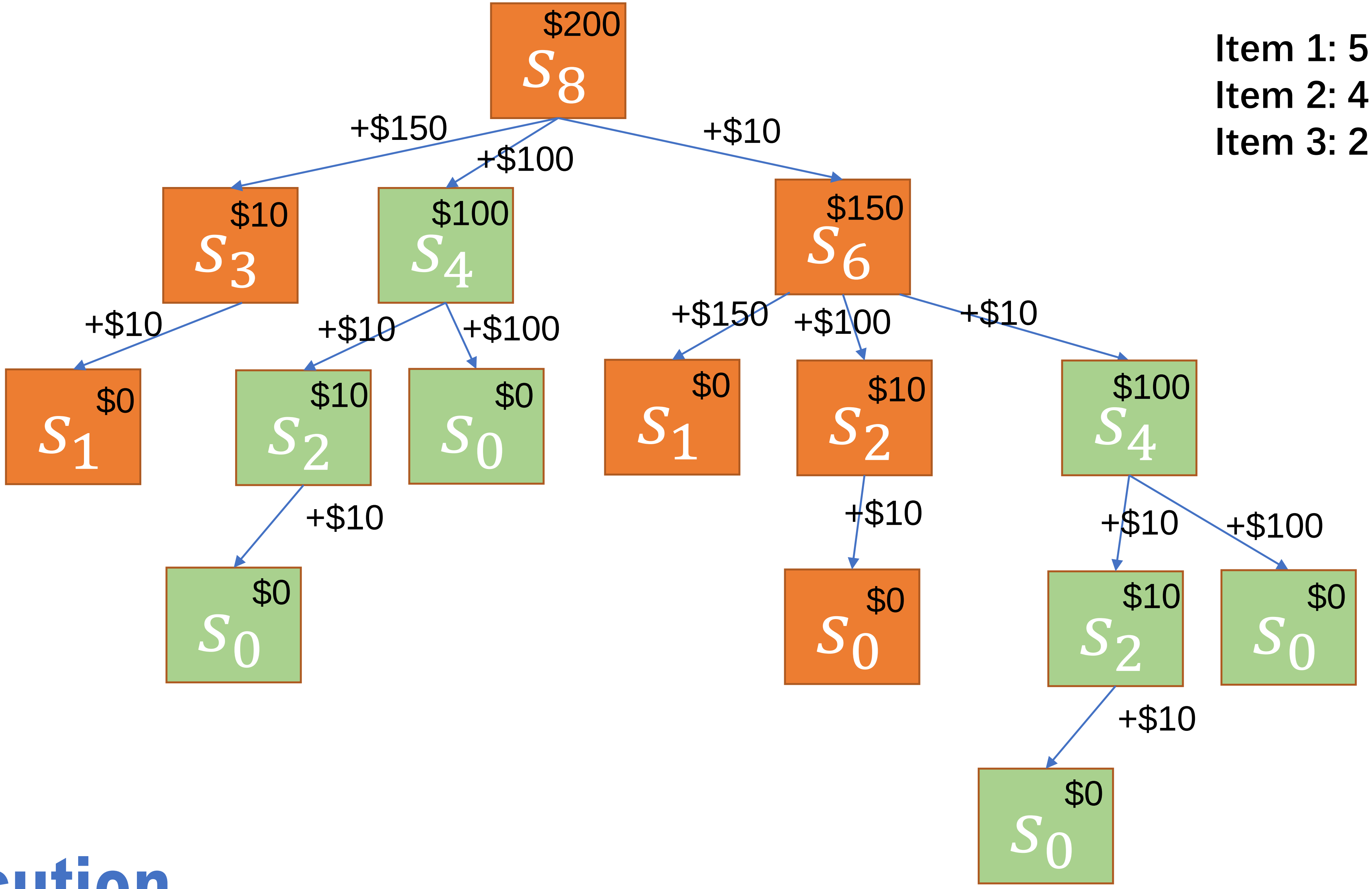
```
answer = suitcase(8);
```

This algorithm takes exponential time, and only works for very small instances



Item 1: 5 lb, $150
Item 2: 4 lb, $100
Item 3: 2 lb, $10

Execution Recurrence Tree



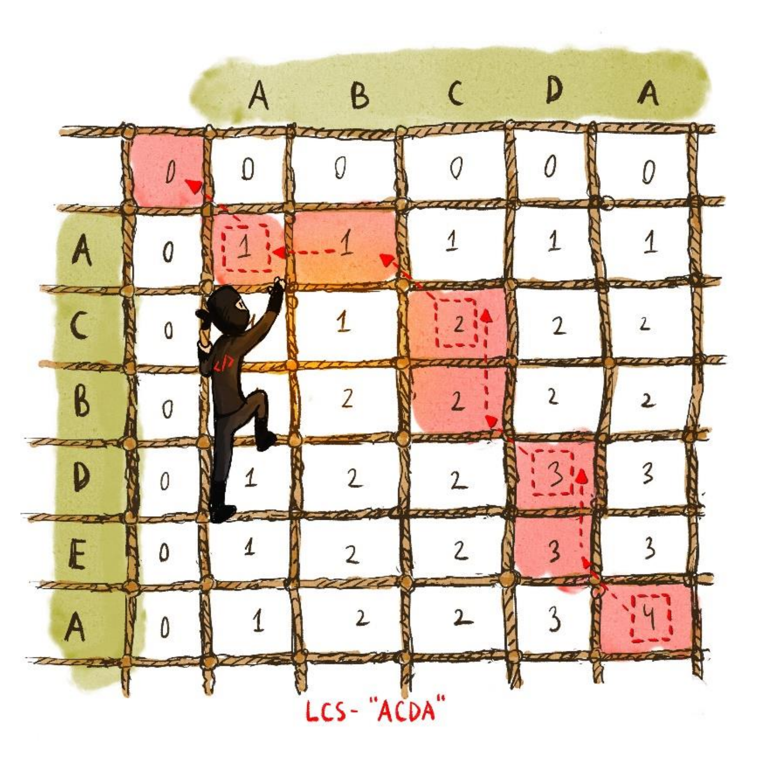
Item 1: 5 lb, $150
Item 2: 4 lb, $100
Item 3: 2 lb, $10

Execution Recurrence Tree

A naïve algorithm

```
int suitcase(int leftWeight) {
    int curBest = 0;
    foreach item (weight, value)
        if (leftWeight >= weight)
            curBest = max(curBest, suitcase(leftWeight - weight) + value);
    return curBest;
}

answer = suitcase(8);
```



Next 3.5 lectures:

Dynamic Programming

Programming?

- **Program (noun) \ ˈprō-ˌgram , -grəm **
  - a sequence of coded instructions that can be inserted into a mechanism (such as a computer)
- **Programming (noun) \ ˈprō-ˌgra-miŋ , -grə-**
  - a plan of action to accomplish a specified end

- **In dynamic programming, or linear programming, the word programming means a "tabular solution method"**
  - In fact, the concept of dynamic programming was proposed before computers, and was a subarea of operating research
  - Without a computer and memory, you have to write down the intermediate results on a piece of paper, and in a table