GREEDY ALGORITHMS

Lecture 12, CMSC 142

Previous Topics

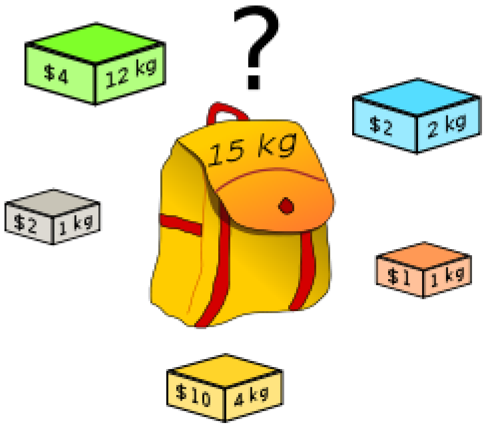
- Introduction to greedy algorithms
- Activity selection

Today’s Topics

- Fractional Knapsack
- Huffman Encoding

Problem

- You are given a knapsack with a limited weight capacity and some items each of which have a weight and a value
- The problem is – “which items to place in the knapsack such that the weight limit is not exceeded and the total value of the items is as large as possible?”



Knapsack Problem Variants

0/1 Knapsack Problem

Items are indivisible i.e
You cannot break an item,
you either take an item or not

Solve using DP

Unbounded Knapsack Problem

Items are divisible i.e.
You can take any fraction of
an item

Solved with a greedy approach

Fractional Knapsack

You may take a whole item or a fraction of the item

Input: n items, each with value and weight, and only one unit of item each (bounded) and knapsack capacity W

Output: Fraction values of each item that maximizes the value

Greedy Approach

- Pick items one by one
- When an item taken as a whole cannot fit in knapsack anymore, take a fraction of the item that will fit in the knapsack and stop

Critical Decision

What is the order of picking items?

1. Decreasing value (Greatest item value first)
2. Increasing size (Smallest weight of item first)
3. Value * Size
4. Value / Size

Decreasing Value

- In what scenario does this fail?

- A heavy item with big value is picked (filling the knapsack) over two lighter items that have a cumulative value greater than other item

Answer

- Ideal item has big value and a small size
- Favor an item with greater value-weight ratio
- Score(item) = value / weight

Intuition

- Get the item with highest value per kilo.
- Even if it is heavy, since it has a high value / kilo, just get all of it.

Question

Given a set of items and knapsack capacity = 60 kg, find the optimal solution, for the fractional knapsack problem by using greedy approach

Items	Weight	Value
1	5	30
2	10	40
3	15	45
4	22	77
5	25	90

Problem

- Find the optimal solution for the fractional knapsack problem by using the greedy approach. Consider

$n = 5$, $w = 60$ kg

$(w_1, w_2, w_3, w_4, w_5) = (5, 10, 15, 22, 25)$

$(v_1, v_2, v_3, v_4, v_5) = (30, 40, 45, 77, 90)$

Problem

A thief enters a house for robbing it. He can carry a maximal weight of 60kg into his bag. There are 5 items in the house with the following weight and value. What items should thief take when he can even take the fraction of any item with him?

Items	Weight	Value
1	5	30
2	10	40
3	15	45
4	22	77
5	25	90

Steps

Step 1. Calculate the ration value/weight for each item

Step 2. Sort the items on the basis of this ratio in decreasing order

Step 3. Starting from the item with the highest ratio, start putting the items into the knapsack. Take as much items as you can.

Solution

Step 1. Calculate the ratio value/weight for each item

Items	Weight	Value	Ratio
1	5	30	6
2	10	40	4
3	15	45	3
4	22	77	3.5
5	25	90	3.6

Solution

Step 2. Sort the items on the basis of the ratio in decreasing.

Items	Weight	Value	Ratio
1	5	30	6
2	10	40	4
5	25	90	3.6
4	22	77	3.5
3	15	45	3

Solution

Step 3. Start putting the items into the knapsack

Items	Weight	Value	Ratio
1	5	30	6
2	10	40	4
5	25	90	3.6
4	22	77	3.5
3	15	45	3

Knapsack Weight	Items in the Knapsack	Cost
60	{}	0

Solution

Step 3. Start putting the items into the knapsack

Items	Weight	Value	Ratio
1	5	30	6
2	10	40	4
5	25	90	3.6
4	22	77	3.5
3	15	45	3

Knapsack Weight	Items in the Knapsack	Cost
60	{}	0
55	1	30

Solution

Step 3. Start putting the items into the knapsack

Items	Weight	Value	Ratio
1	5	30	6
2	10	40	4
5	25	90	3.6
4	22	77	3.5
3	15	45	3

Knapsack Weight	Items in the Knapsack	Cost
60	{}	0
55	1	30
45	1, 2	70

Solution

Step 3. Start putting the items into the knapsack

Items	Weight	Value	Ratio
1	5	30	6
2	10	40	4
5	25	90	3.6
4	22	77	3.5
3	15	45	3

Knapsack Weight	Items in the Knapsack	Cost
60	{}	0
55	1	30
45	1, 2	70
20	1, 2, 5	160

Solution

Now, knapsack weight left is 20kg but item 4 has a weight of 22kg. Had the problem been a 0/1 knapsack problem, we would have stopped and said that knapsack has items {1, 2, 5} and the total cost of the knapsack is 160.

For a fractional knapsack problem, knapsack contains the items –

{1, 2, 5, $\frac{20}{22}T4$}

Total weight = 5+10+25+ $\frac{(60-40)}{22}22$=60

and the total cost = 160 + $\frac{20}{22}77$

= 230

Analysis

Sorting: O(nlogn)

While loop: O(n)

RT: O(nlogn)

Huffman Encoding

Text Compression

Different than data compression in general

- On a computer: changing the representation of a file so that it takes less space to store or/and less time to transmit.

Example

n xmpl f lssy lgrthm fr cmprssng txt wld b t rmv ll th vwls.

An example of a lossy algorithm for compressing text would be to remove all the vowels.”

Huffman Encoding

Computer Data Encoding:

How do we represent data in binary?

Historical Solution:

Fixed length codes.

Encode every symbol by a unique binary string of a fixed length.

Example: ASCII (7 bit code)

ASCII Example:

ASCII BINARY ALPHABET

A	1000001	N	1001110
B	1000010	O	1001111
C	1000011	P	1010000
D	1000100	Q	1010001
E	1000101	R	1010010
F	1000110	S	1010011
G	1000111	T	1010100
H	1001000	U	1010101
I	1001001	V	1010110
J	1001010	W	1010111
K	1001011	X	1011000
L	1001100	Y	1011001
M	1001101	Z	1011010

AABCAA

A	A	B	C	A	A
1000001	1000001	1000010	1000011	1000001	1000001

Total space usage in bits:

Assume an ℓ bit fixed length code.

For a file of n characters

Need $n\ell$ bits.

Can we do better than this?

Variable Length codes

Idea:

In order to save space, use fewer bits for frequent characters and more bits for rare characters.

Example:

Suppose alphabet of 3 symbols: { A, B, C }.

Suppose in file: 1,000,000 characters.

Need 2 bits for a fixed length code for a total of 2,000,000 bits.

Variable Length codes - example

Suppose the frequency distribution of the characters is:

A	B	C
999,000	500	500

Encode:

A	B	C
0	10	11

Note that the code of A is of length 1, and the codes for B and C are of length 2

Total space usage in bits:

Fixed code: 1,000,000 x 2 = 2,000,000

Varable code: 999,000 x 1

\+ 500 x 2

500 x 2

1,001,000

A savings of almost 50%

How do we decode?

In the fixed length, we know where every character starts, since they all have the same number of bits.

Example: A = 00
B = 01
C = 10

A	A	A	B	B	C	C	C	B	C	B	A	A	C	C
00	00	00	01	01	10	10	10	01	10	01	00	00	10	10

How do we decode?

In the variable length code, we use an idea called Prefix code, where no code is a prefix of another.

Example: A = 0
B = 10
C = 11

None of the above codes is a prefix of another.
We could not encode A as 0 and B as 01, since 0 is a prefix of 01.

How do we decode?

Example: A = 0

B = 10

C = 11

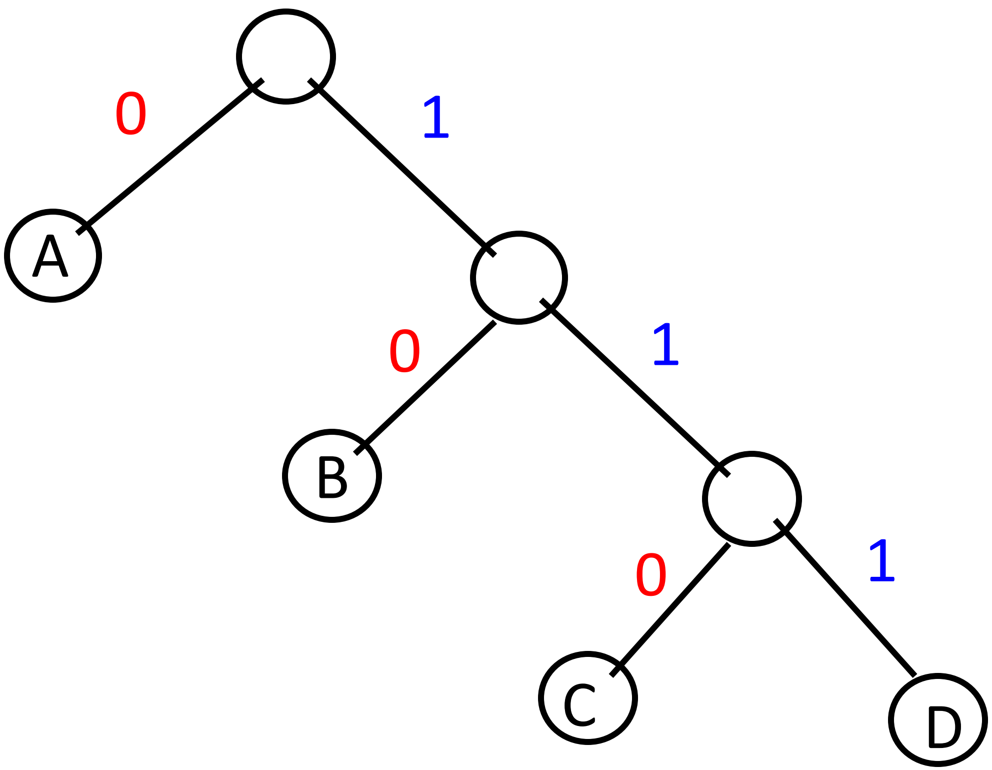
So, for the string:

A	A	A	B	B	C	C	C	B	C	B	A	A	C	C
0	0	0	10	10	11	11	11	10	11	10	0	0	0	0

Idea

Consider a binary tree, with:

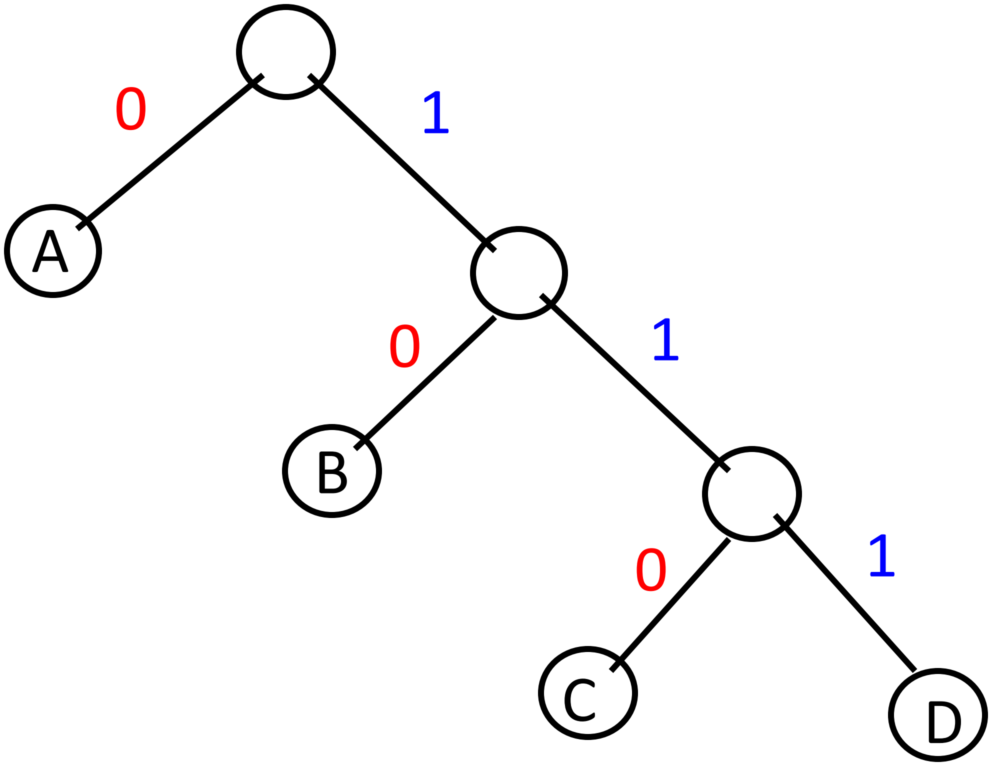
- 0 meaning a left turn
- 1 meaning a right turn.



Idea

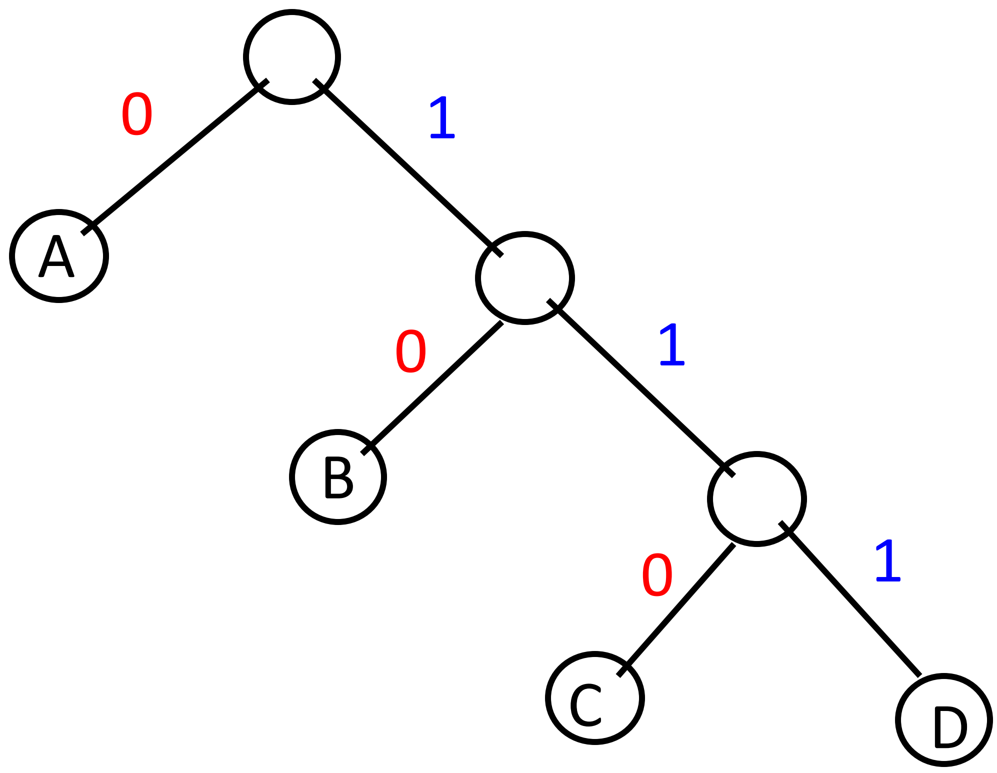
Consider the paths from the root to each of the leaves A, B, C, D:

A : 0
B : 10
C : 110
D : 111



Observe:

1. This is a prefix code, since each of the leaves has a path ending in it, without continuation.
2. If the tree is full then we are not “wasting” bits.
3. If we make sure that the more frequent symbols are closer to the root then they will have a smaller code.



Greedy Algorithm:

1. Consider all pairs: <frequency, symbol>.

2. Choose the two lowest frequencies, and make them brothers, with the root having the combined frequency.

3. Iterate.

Greedy Algorithm Example:

Alphabet: A, B, C, D, E, F

Frequency table:

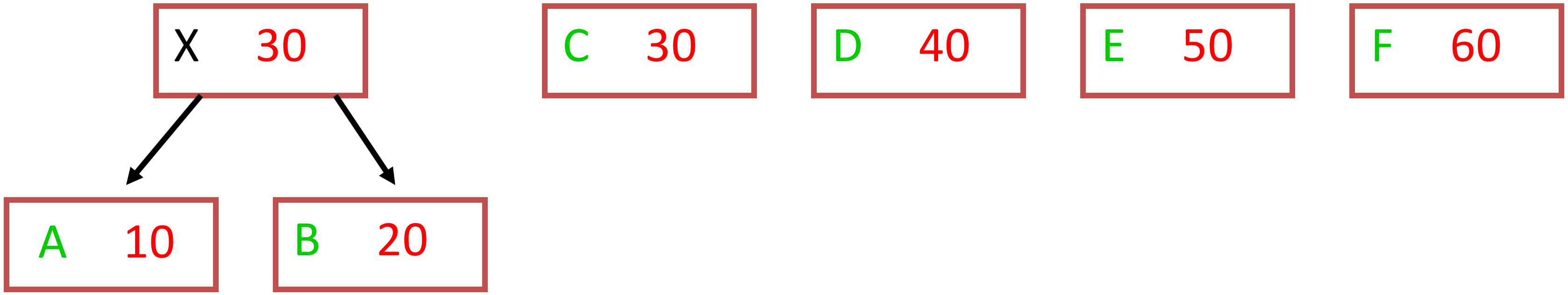
A	B	C	D	E	F
10	20	30	40	50	60

Total File Length: 210

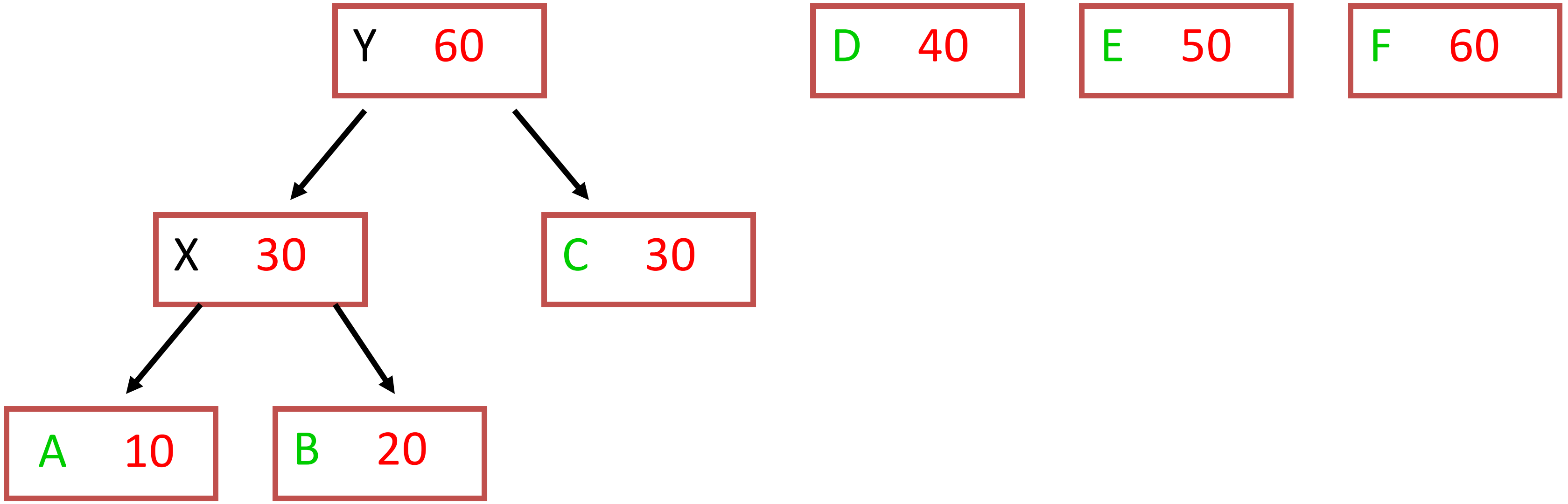
Algorithm Run:

A 10	B 20	C 30	D 40	E 50	F 60

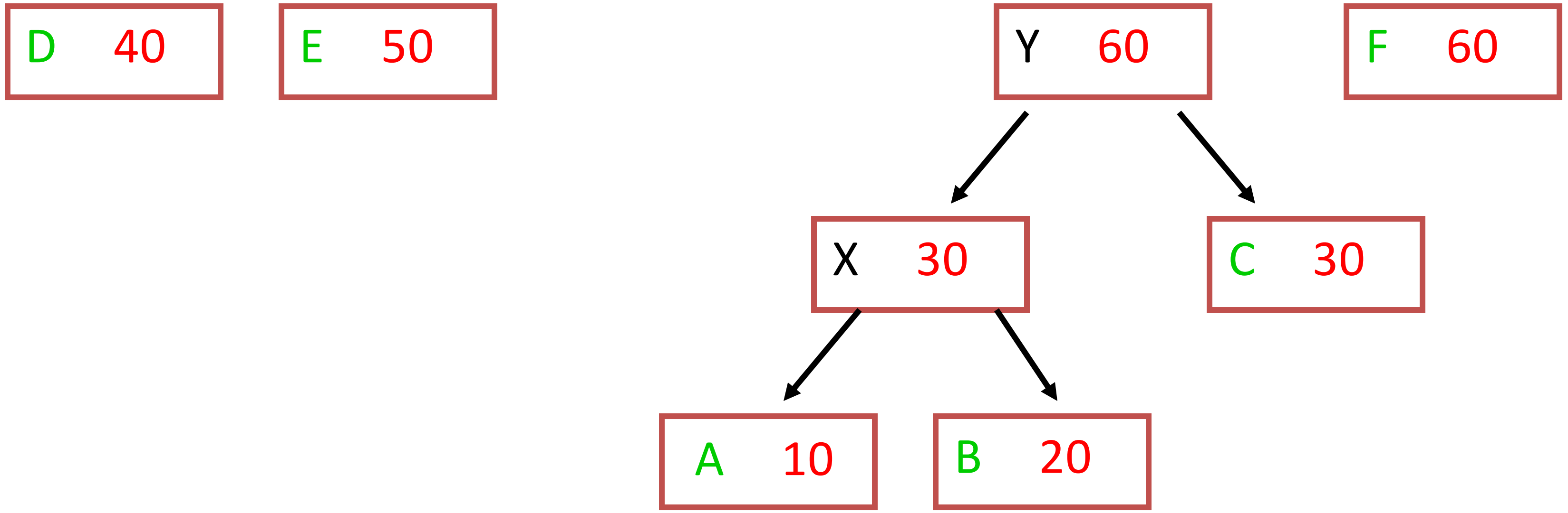
Algorithm Run:



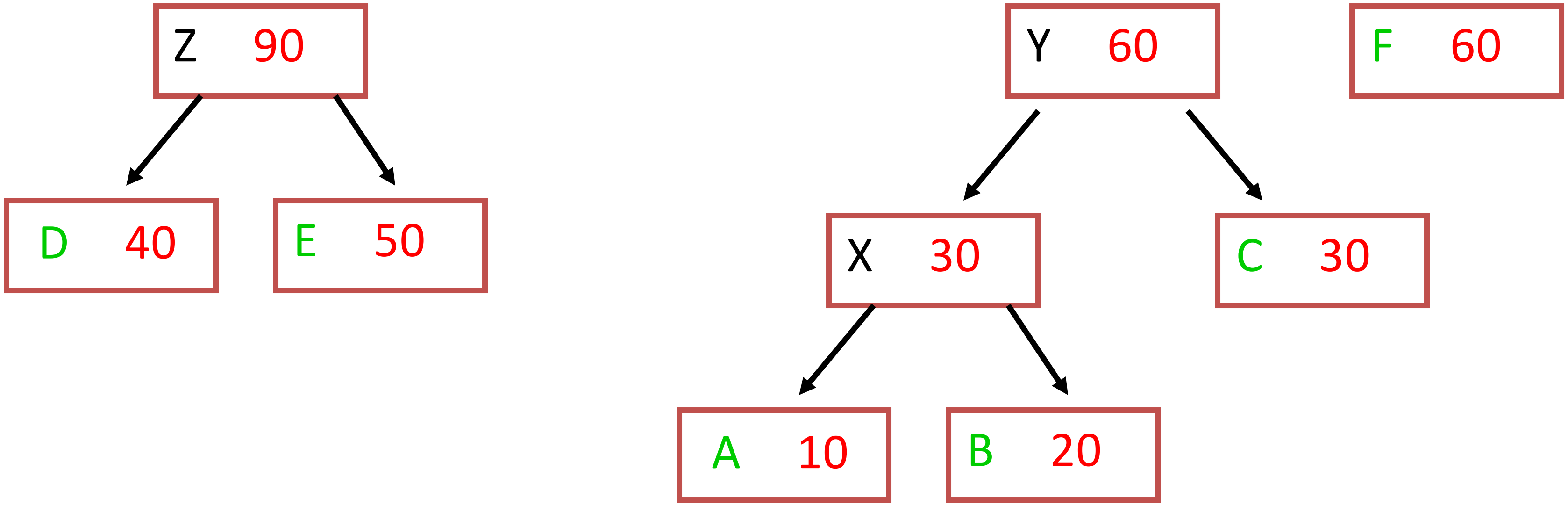
Algorithm Run:



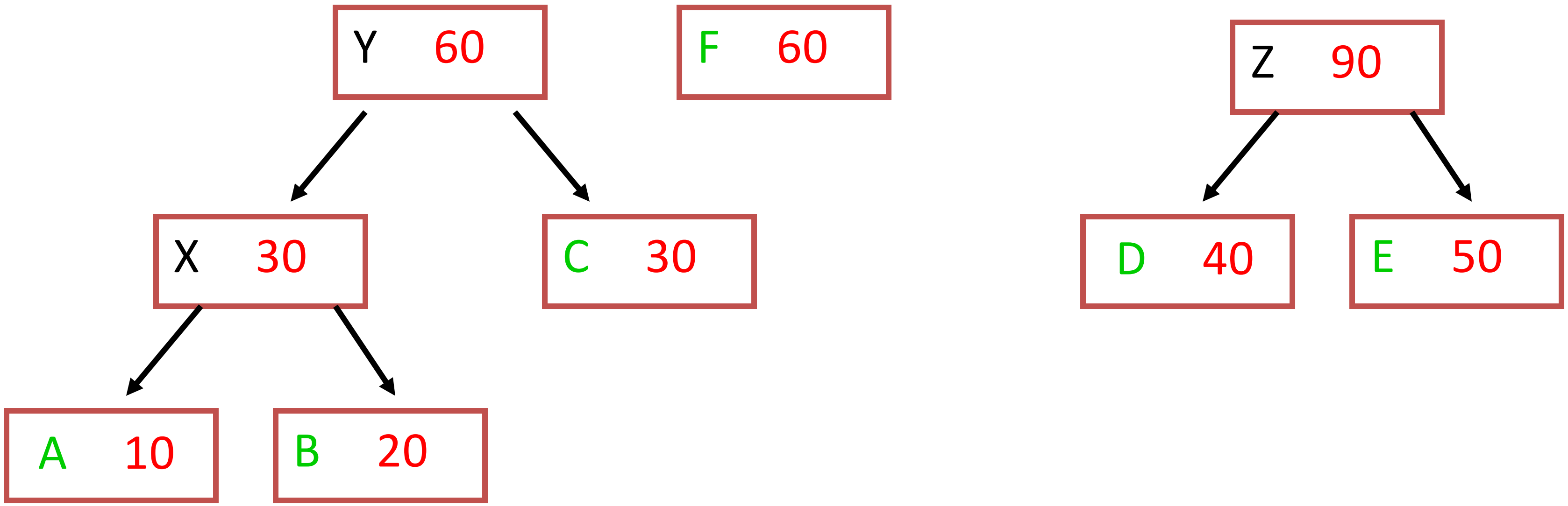
Algorithm Run:



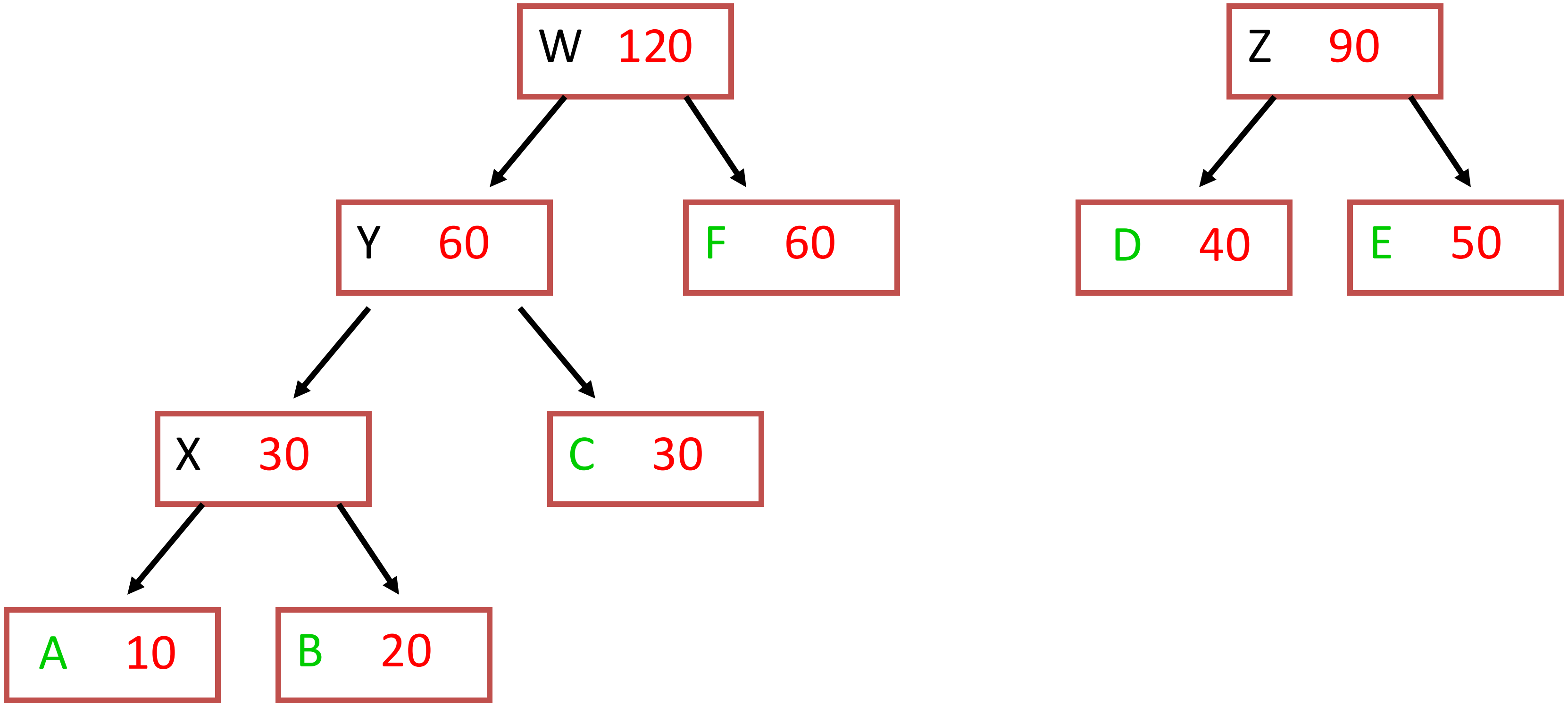
Algorithm Run:



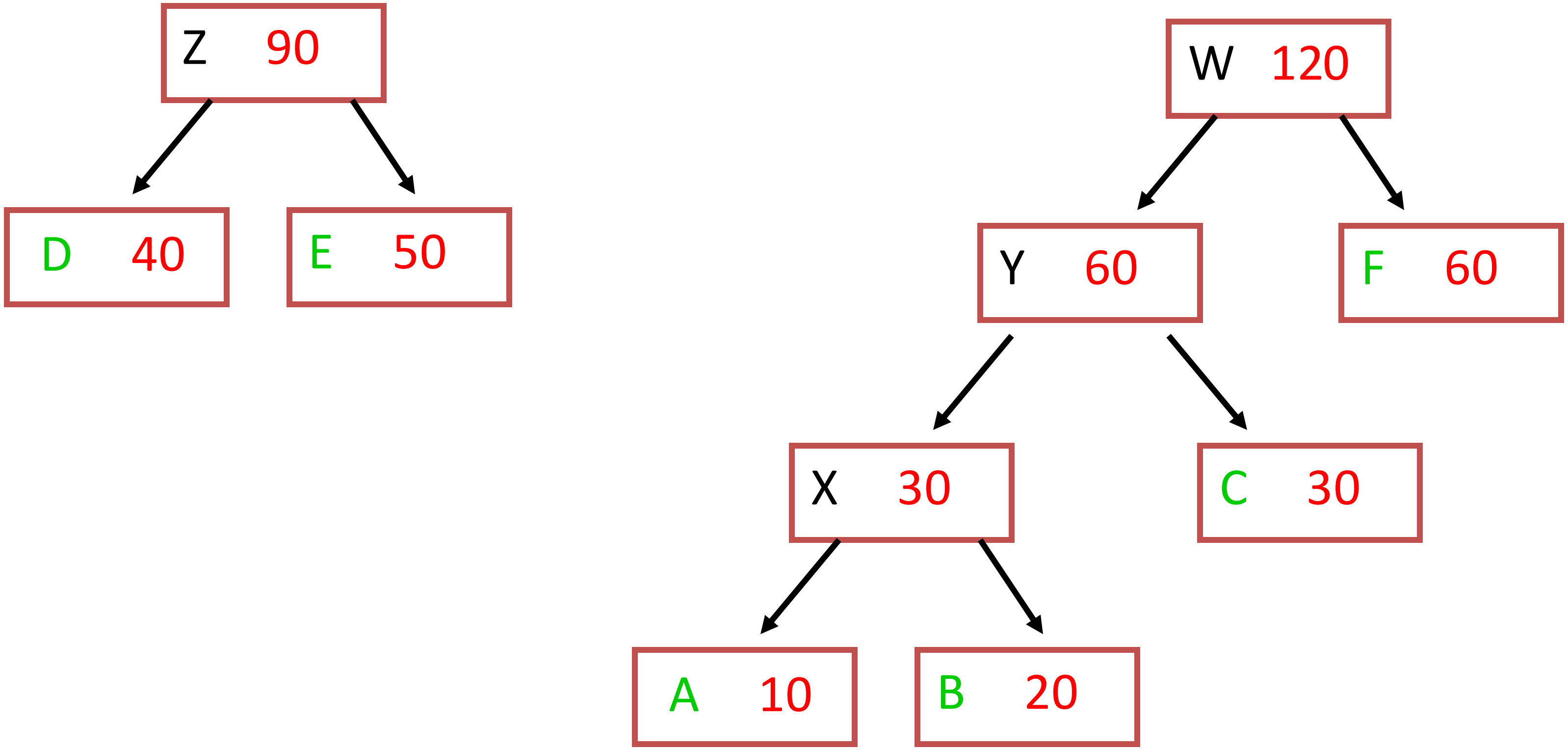
Algorithm Run:



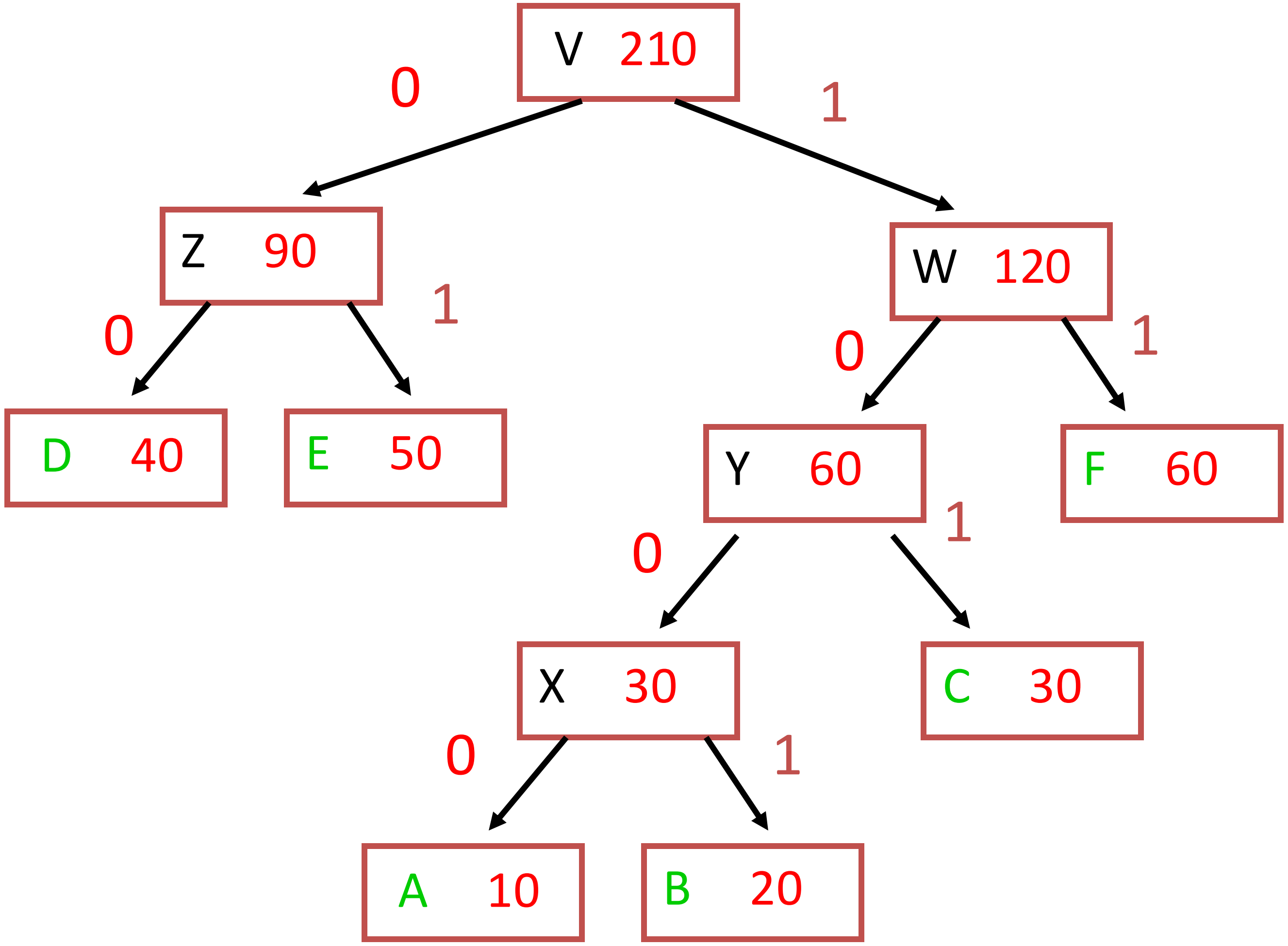
Algorithm Run:



Algorithm Run:

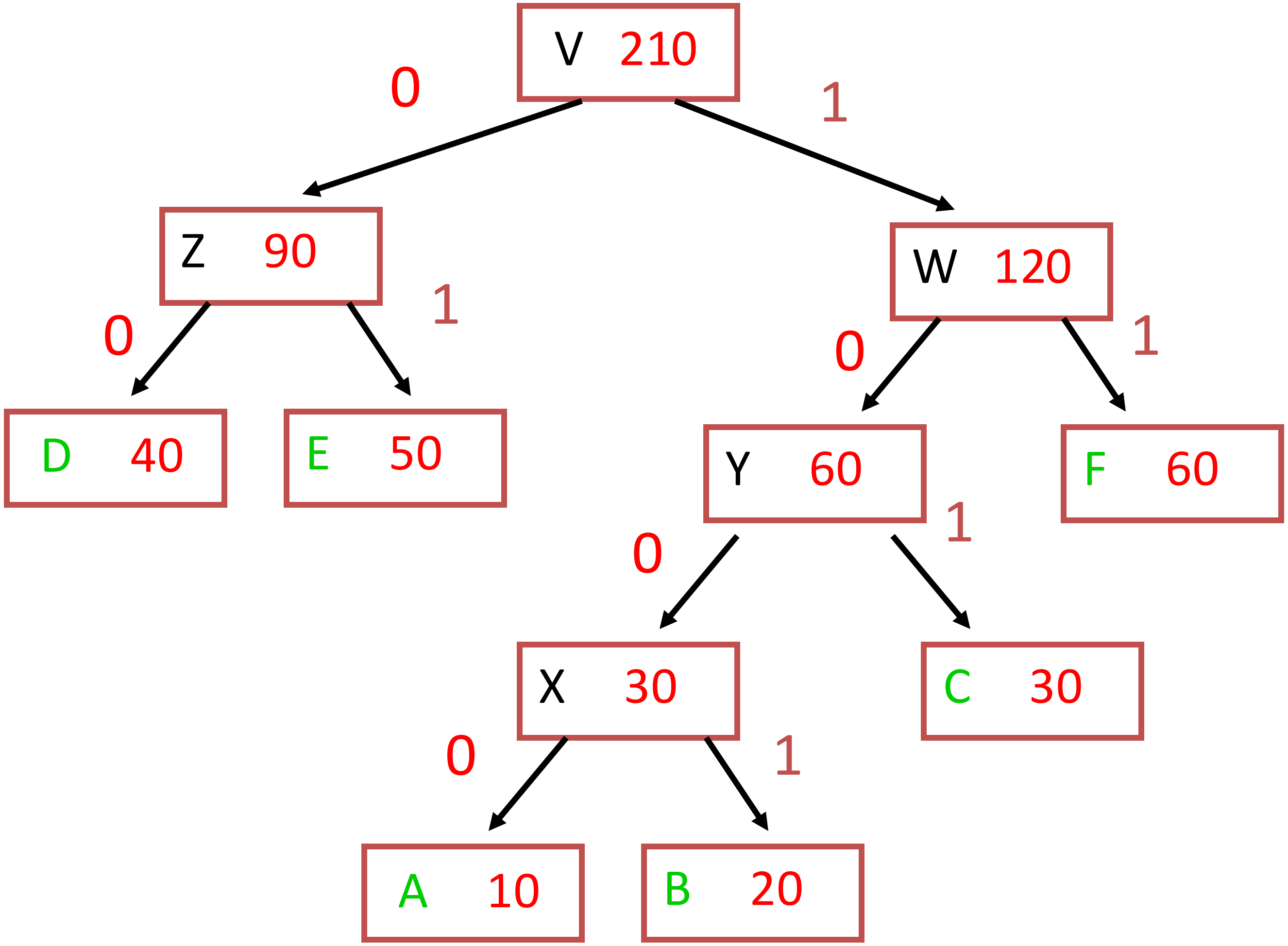


Algorithm Run:



Algorithm Run:

A: 1000
B: 1001
C: 101
D: 00
E: 01
F: 11



File Size: 10x4 + 20x4 + 30x3 + 40x2 + 50x2 + 60x2 =
40 + 80 + 90 + 80 + 100 + 120 = 510 bits

Note the savings:

The Huffman code:

Required 510 bits for the file.

Fixed length code:

Need 3 bits for 6 characters.

File has 210 characters.

Total: 630 bits for the file.

Formally, the algorithm:

Initialize trees of a single node each.

Keep the roots of all subtrees in a priority queue.

Iterate until only one tree left:

Merge the two smallest frequency subtrees into a single subtree with two children, and insert into priority queue.

Algorithm

Algorithm 1 HUFFMAN(C)

$n := |C|$;
$Q := C$;
for $i := 1$ **to** $n - 1$ **do**
    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);
end for
return EXTRACT-MIN(Q); {return the root of the tree}

Algorithm time:

Each priority queue operation (e.g. heap):

$O(\log n)$

In each iteration: one less subtree.

Initially: n subtrees.

Total: $O(n \log n)$ time.

End of Lecture