Greedy Algorithm

Greedy algorithm is an algorithm technique which always takes the best immediate, or local, solution while finding an answer.

It is a mathematical process that looks for simple, easy-to-implement solutions to complex, multi-step problems by deciding which next step will provide the most obvious benefit (perhaps why it is called greedy).

Greedy is a strategy to solve the complex problem or you can say optimization problem in simple and quicker way. This strategy has following to characteristics

1. **Greedy-choice property: A global optimum can be arrived at by selecting a local optimum.**

2. Optimal substructure: An optimal solution to the problem contains an optimal solution to sub problems.

Below is the list of famous greedy algorithms:

-Prism's algorithm
-Kruskal algorithm
-Reverse-Delete algorithm
-Dijkstra's algorithm
-Huffman coding

EXAMPLE PROBLEM : ACTIVITY SELECTION PROBLEM (Activity selection problem is an example of greedy algorithm)

An activity-selection is the problem of scheduling a resource among several competing activity. Given M activities with their start and finish times,we have to find the maximum number of activities that can be performed by a single person,assuming that a person can only work on a single activity at a time.

GREEDY ALGO STEPS

a) Sort the activities according to their finishing time

b) Select the first activity from the sorted array and print it.

c) Do following for remaining activities in the sorted array.

d) If the start time of this activity is greater than the finish time of previously selected activity then select this activity and print it.

Consider the following 6 activities.

s_time[] = {1, 3, 0, 5, 3, 5, 6, 8, 8, 2}; // start time

f_time[] = {4, 5, 6, 7, 9, 9, 10, 11, 12, 18}; // finish time ,

```
#include
void Activities(int s_time[], int f_time[], int n)

{

int i, j;

printf ("Selected Activities are:");

i = 1; printf("A%d ", i);

for (j = 1; j < n; j++) {

if (s_time[j] >= f_time[i]) {

printf ("A%d ", j+1); i = j;

}

}

}

int main() {

int s_time[] = {1, 3, 0, 5, 3, 5, 6, 8, 8, 2};

int f_time[] = {4, 5, 6, 7, 9, 9, 10, 11, 12, 18};

int n = sizeof(s_time)/sizeof(s_time[0]);

Activities(s, f, n);

return 0;

}
```

Example (that works) - Huffman code

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.

Examples: ASCII (7 bit code),

EBCDIC (8 bit code), …

American Standard Code for Information Interchange

b_7 →						0	0	0	0	1	1	1	1
b_6 →						0	0	1	1	0	0	1	1
b_5 →						0	1	0	1	0	1	0	1
Bits	b_4 ↓	b_3 ↓	b_2 ↓	b_1 ↓	Column → / Row ↓	0	1	2	3	4	5	6	7
	0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
	0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
	0	0	1	0	2	STX	DC2	"	2	B	R	b	r
	0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
	0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
	0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
	0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
	0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
	1	0	0	0	8	BS	CAN	(	8	H	X	h	x
	1	0	0	1	9	HT	EM	)	9	I	Y	i	y
	1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
	1	0	1	1	11	VT	ESC	+	;	K	[	k	{
	1	1	0	0	12	FF	FC	,	<	L	\	l	|
	1	1	0	1	13	CR	GS	-	=	M	]	m	}
	1	1	1	0	14	SO	RS	.	>	N	^	n	~
	1	1	1	1	15	SI	US	/	?	O	_	o	DEL

ASCII Example:

b_7					0	0	0	0	1	1	1	1
b_6					0	0	1	1	0	0	1	1
b_5					0	1	0	1	0	1	0	1
Bits b_4	b_3	b_2	b_1	Column → / Row ↓	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FC	,	<	L	\	l	|
1	1	0	1	13	CR	GS	-	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

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.

Variable Length codes

Idea: In order to save space, use less 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

000000010110101001100100001010

A A A B B C C C B C B A A C C

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.

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 the encoding:

0 0 010101111111101110 0 01111

Prefix Code

Example: A = 0
B = 10
C = 11

Decode the string

0|0 |0|10|10|11|11|11|10|11|10|0 |0|11|11 |

A A A B B C C C B C B A A C C

Desiderata:

Construct a variable length code for a given file with the following properties:

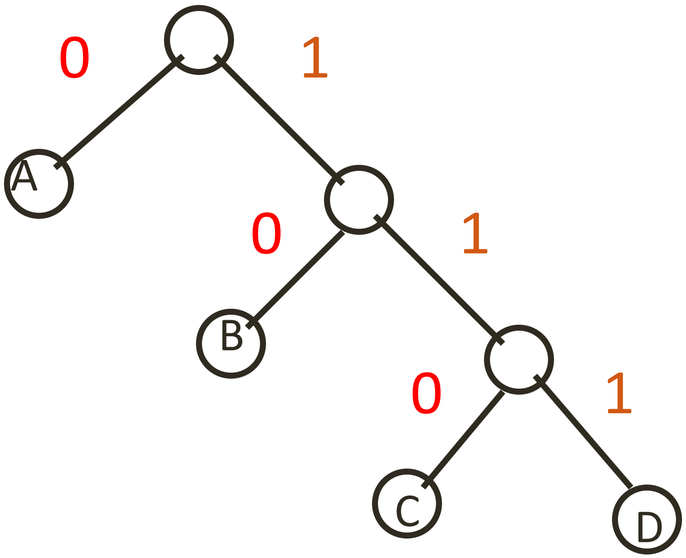
1. Prefix code.
2. Using shortest possible codes.
3. Efficient.
4. As close to entropy as possible.

Idea

Consider a binary tree, with:

0 meaning a left turn

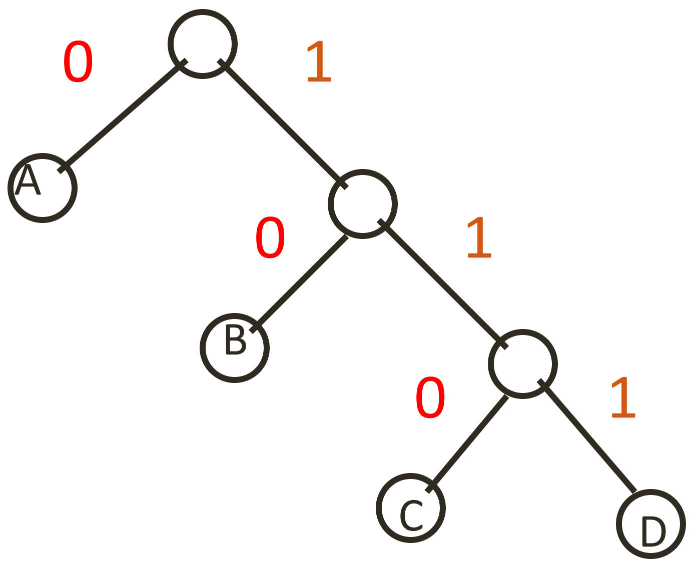
1 meaning a right turn.



Idea

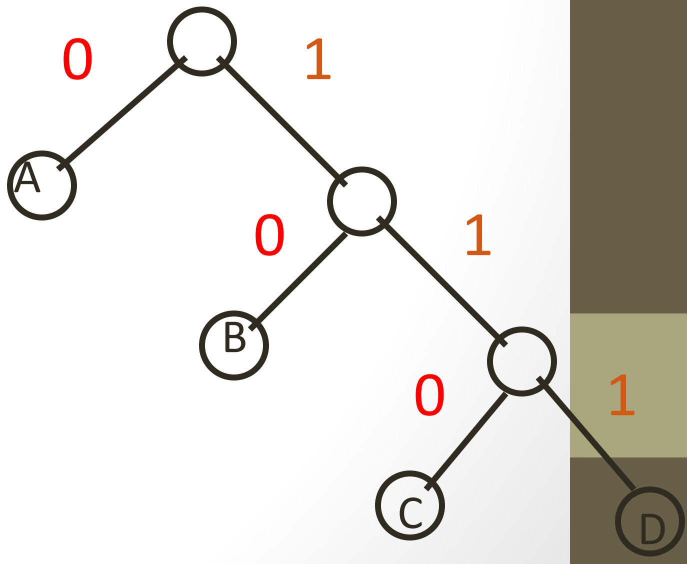
Consider the paths from the root to each of the leaves A, B, C, D:

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:

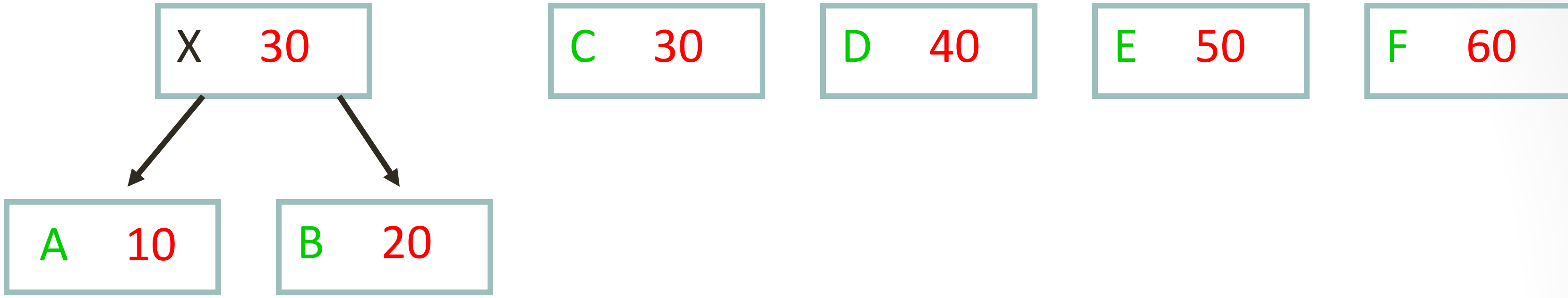
A	B	C	D	E	F
10	20	30	40	50	60

Total File Length: 210

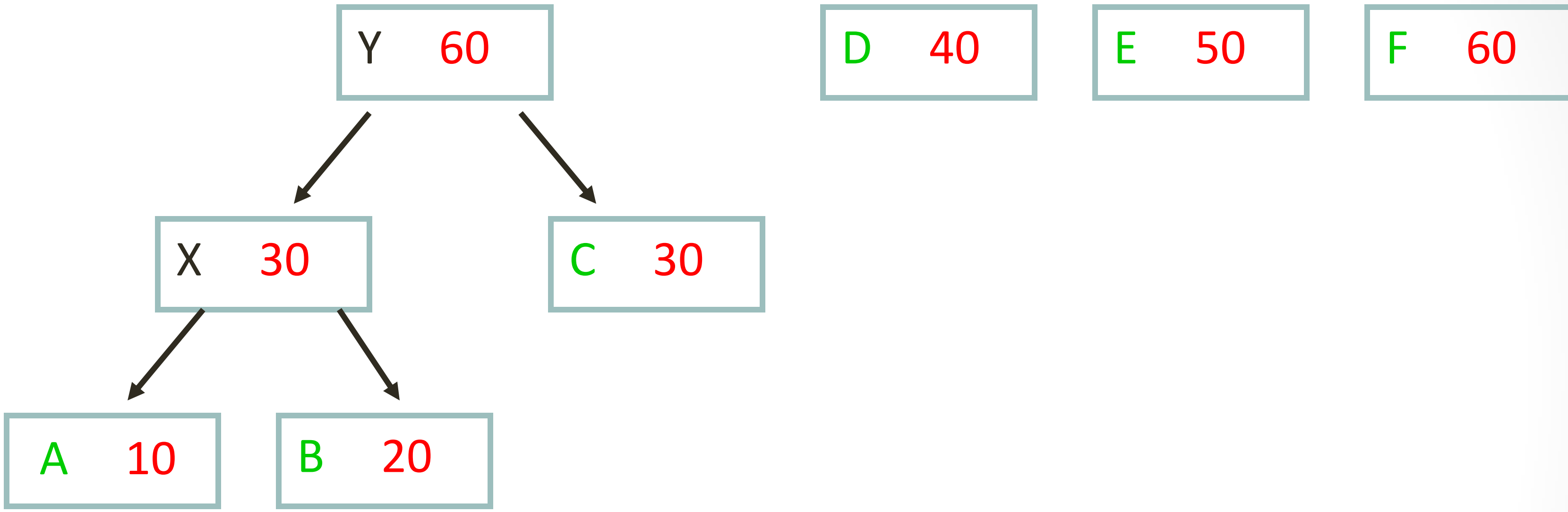
Algorithm Run:

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

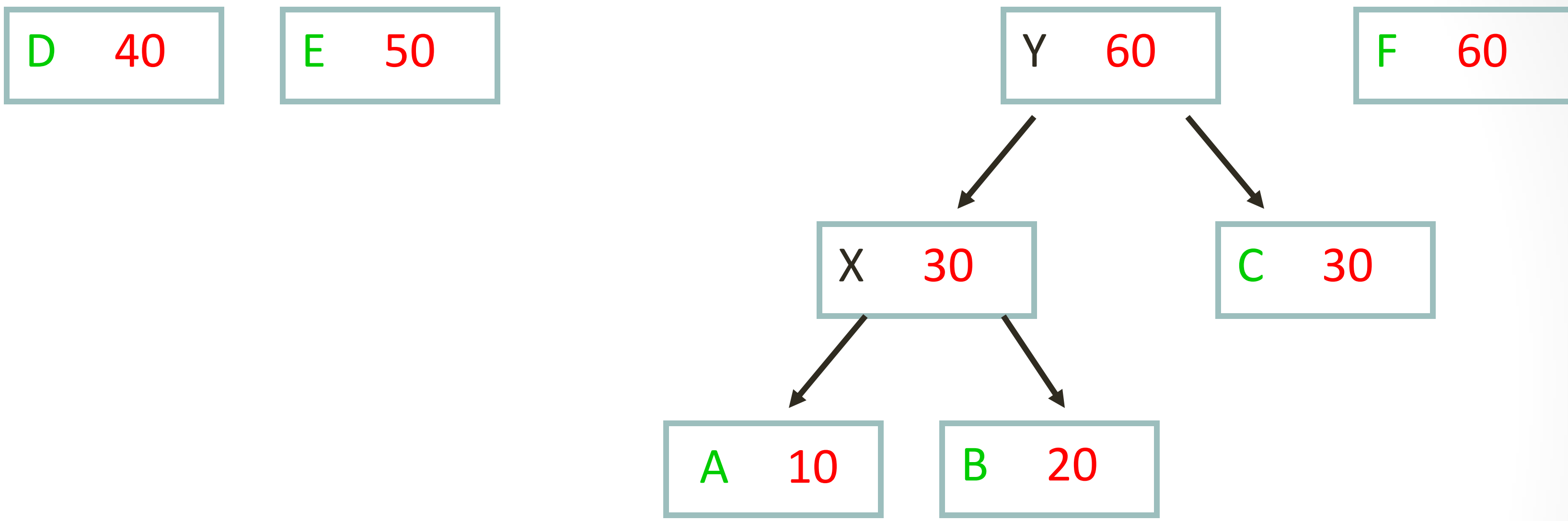
Algorithm Run:



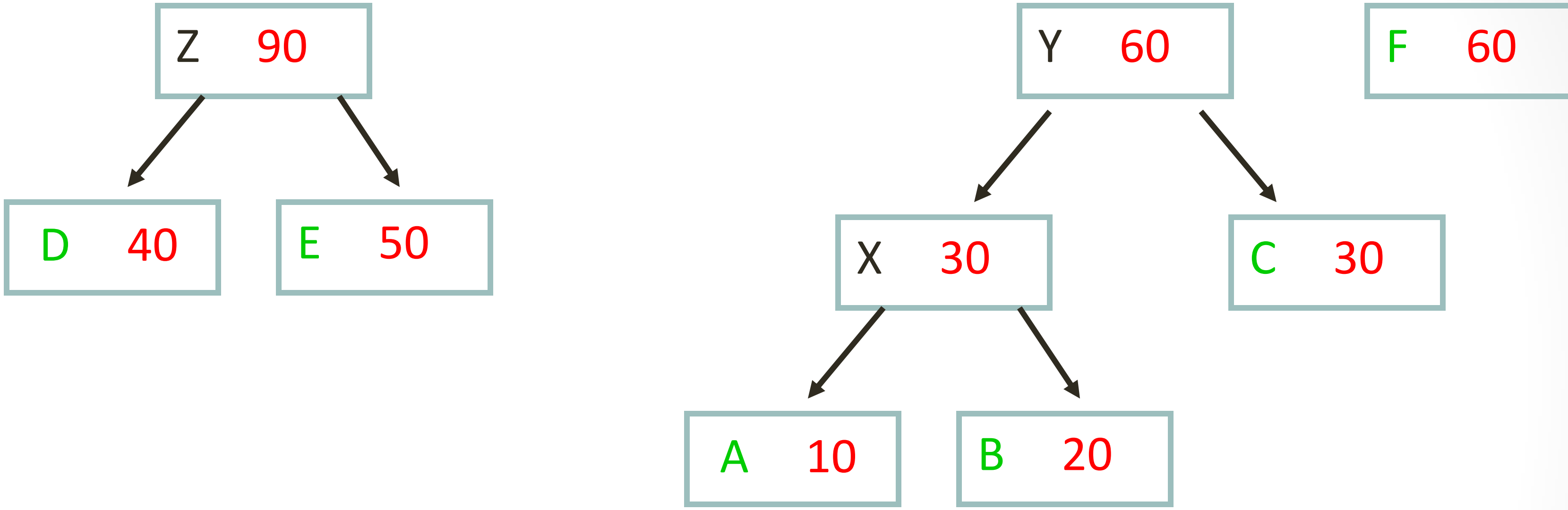
Algorithm Run:



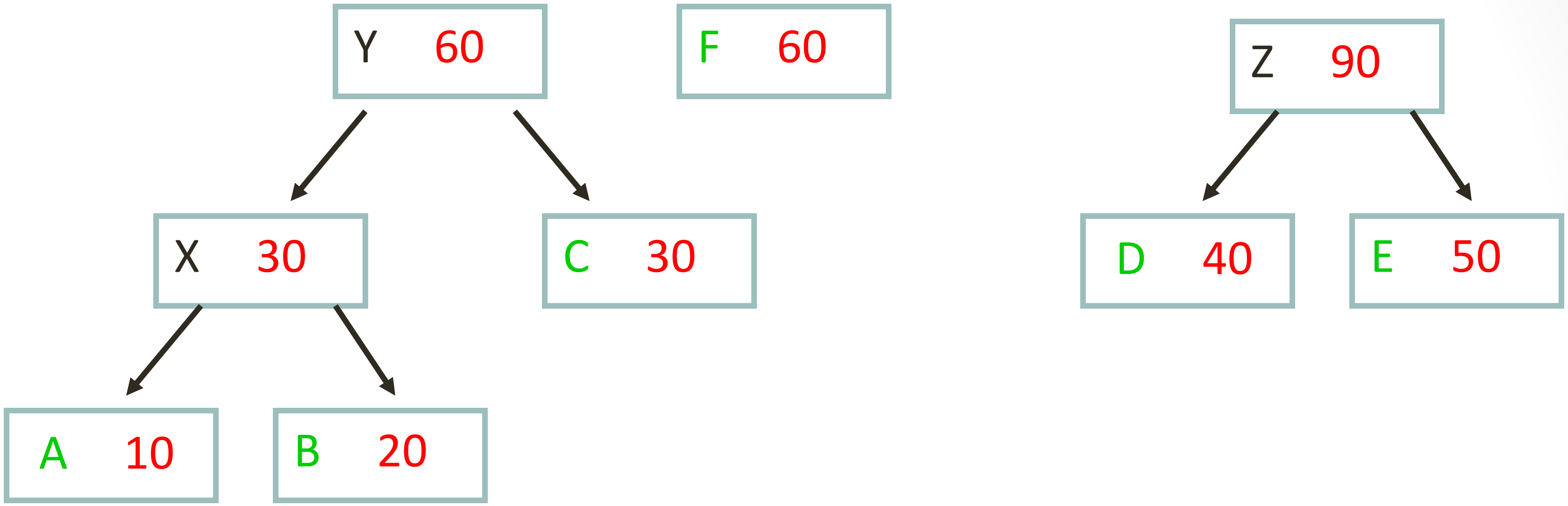
Algorithm Run:



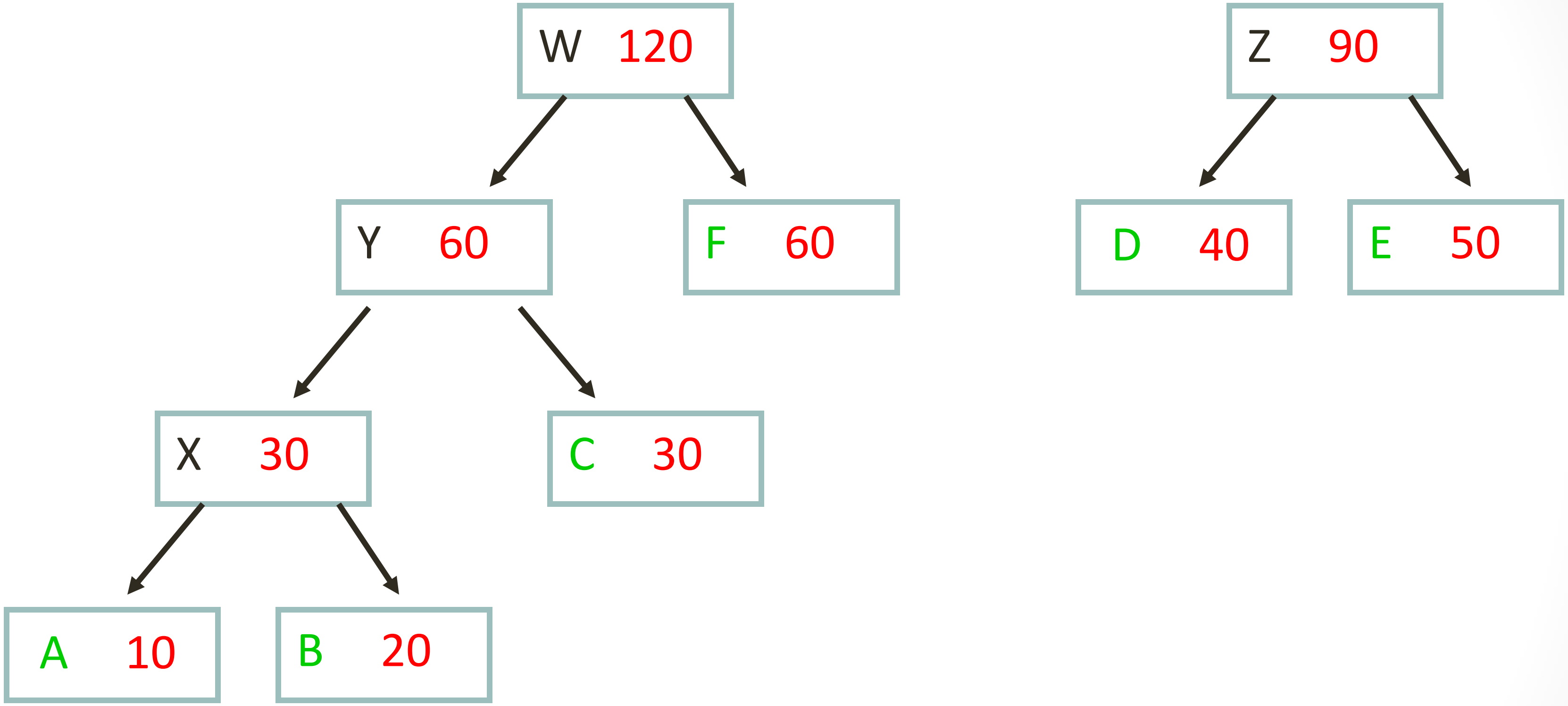
Algorithm Run:



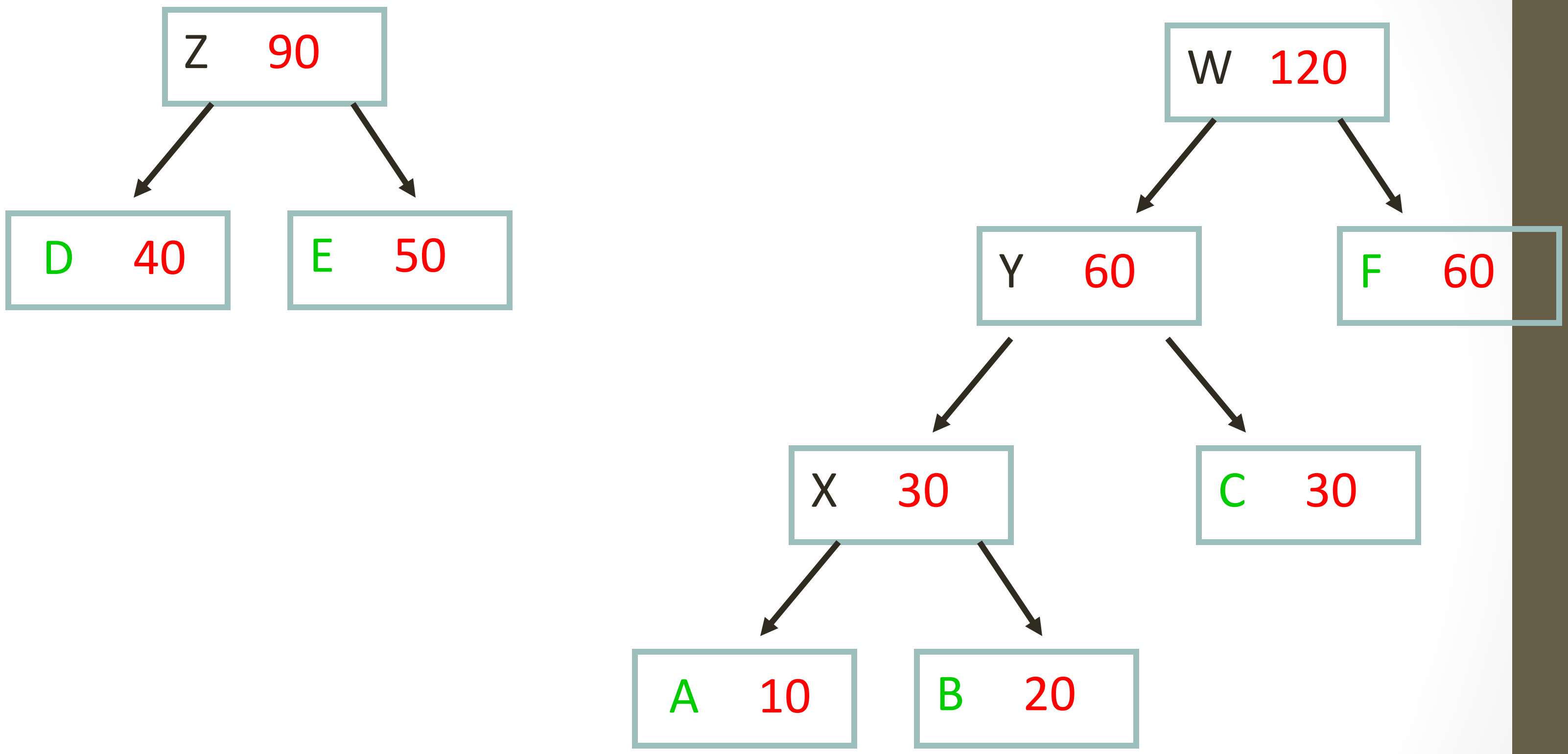
Algorithm Run:



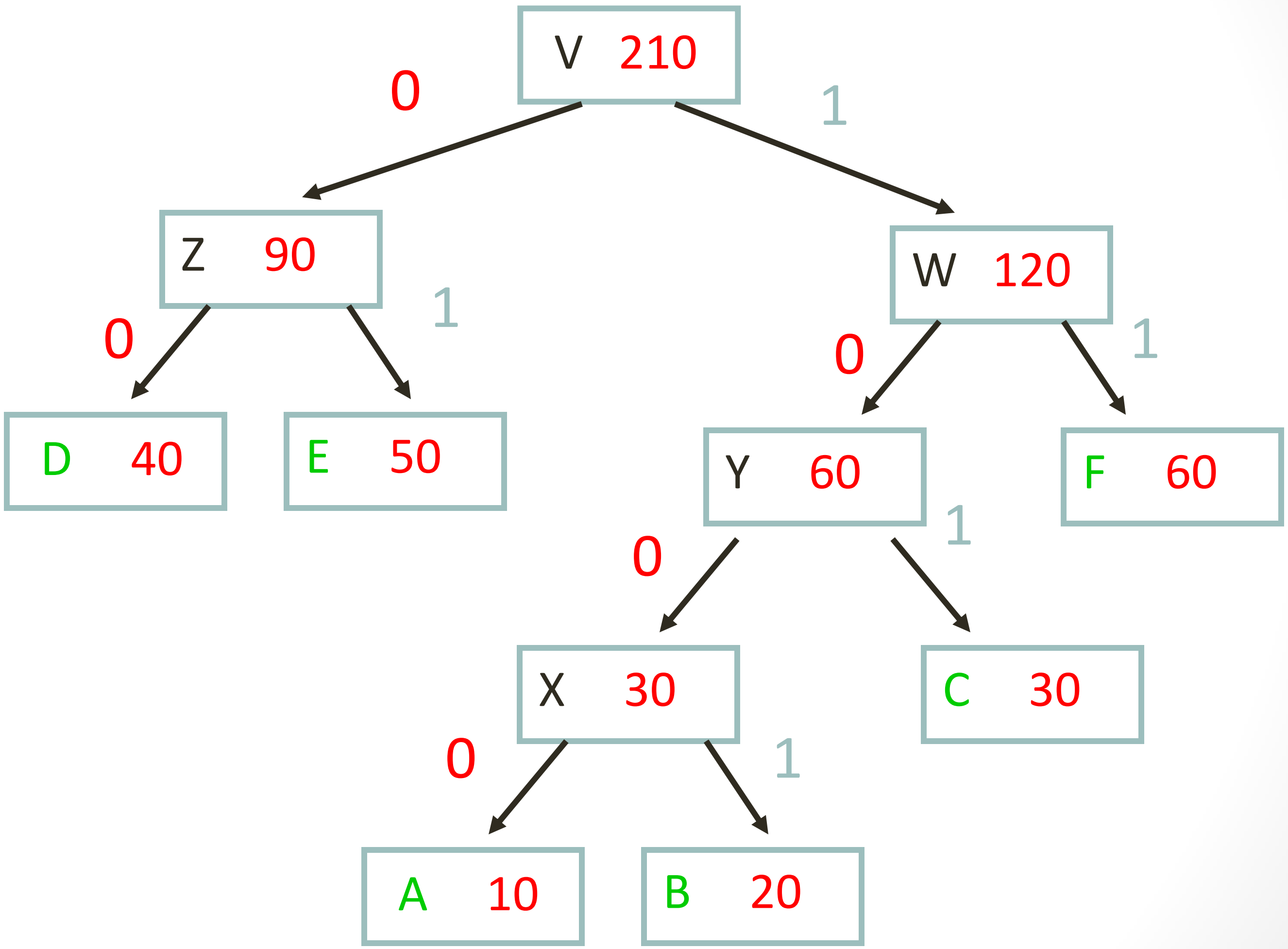
Algorithm Run:



Algorithm Run:

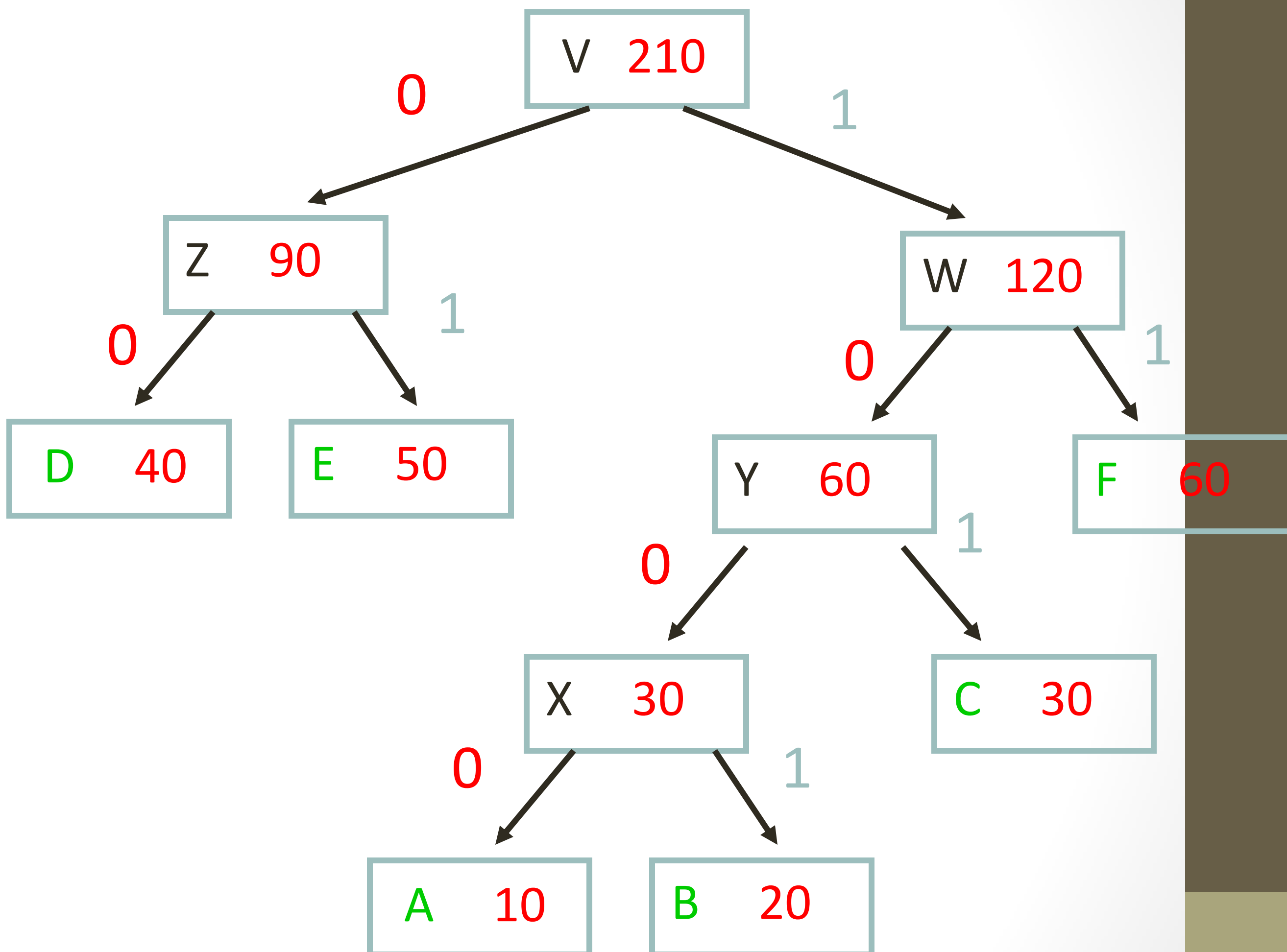


Algorithm Run:



The Huffman encoding:

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.

Note also:

For uniform character distribution:

The Huffman encoding will be equal to the fixed length encoding.

Why?

Assignment.