ECE 264 Spring 2023

Advanced C Programming

Aravind Machiry
Purdue University

Midterm 3

- When: Thursday (6th April)
- How: Online via Brightspace for 24 hours from 7:30 am (9th) to 7:29 am (10th)
- Time : 3 hours (Expected to be done in 1 hour).
- Questions similar to quiz but expect some code to be understood or written.

Topics for Midterm 3

- Linked Lists
- Binary Trees
- Recursion and Shuffling
- Some questions related to expressions

Linked lists

- Reasoning about code:
  - while (p != NULL) {

    ….

    p = p->next;

    }
  - What happens if I remove the line “p = p->next” ?

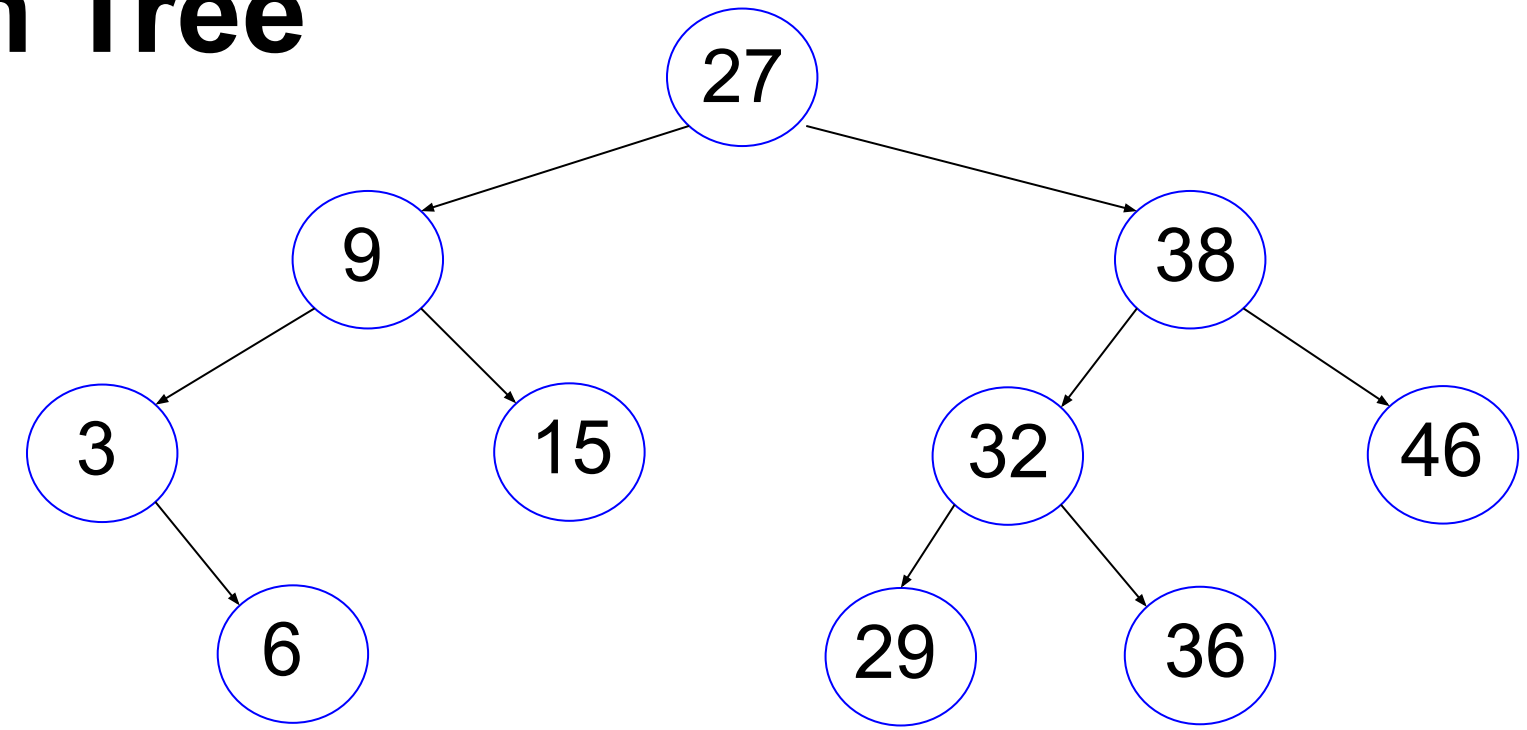
- Use after free -- free only after you are done with a node?

Binary Trees

- Structure of a tree?

- Number of nodes in a complete binary tree.

Binary Search Tree

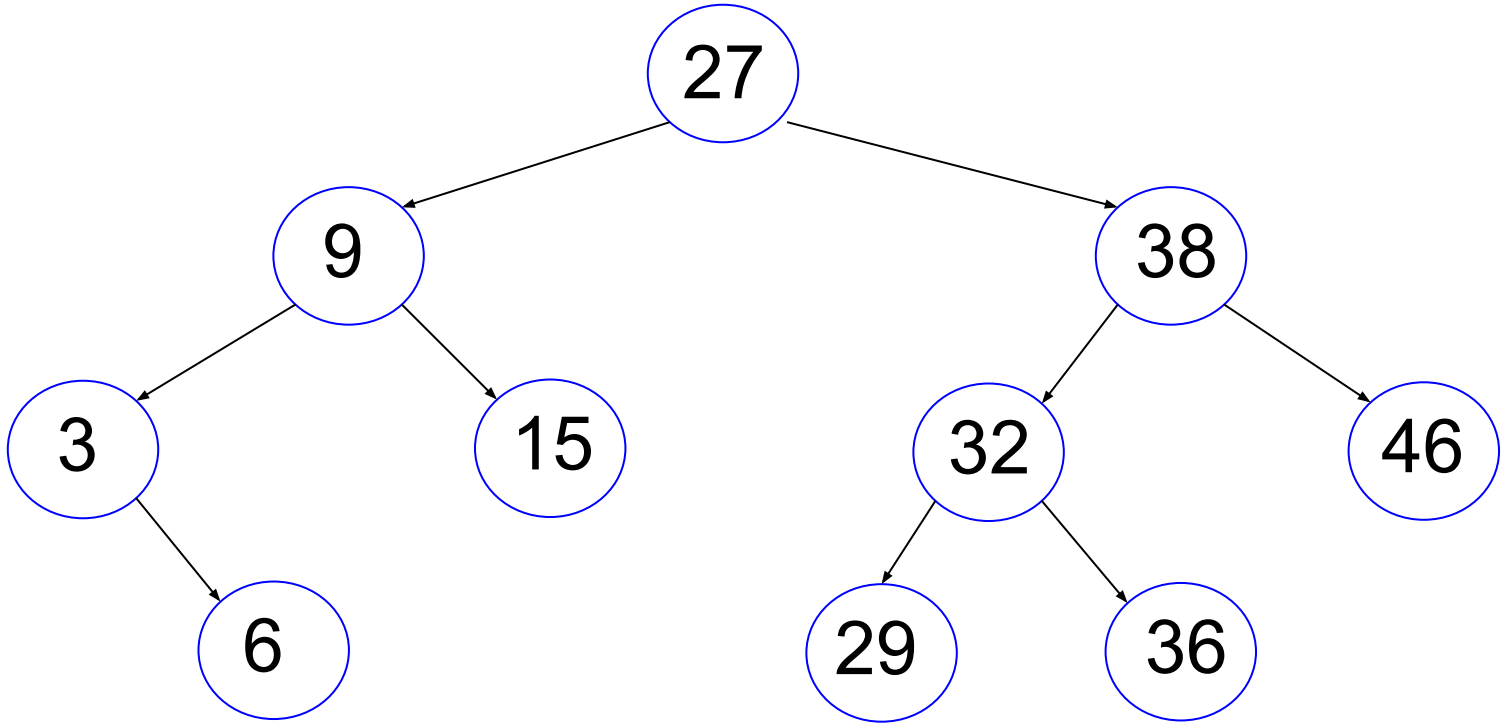


yunglu@purdue.edu

Binary Search Tree

How to create a tree like this?

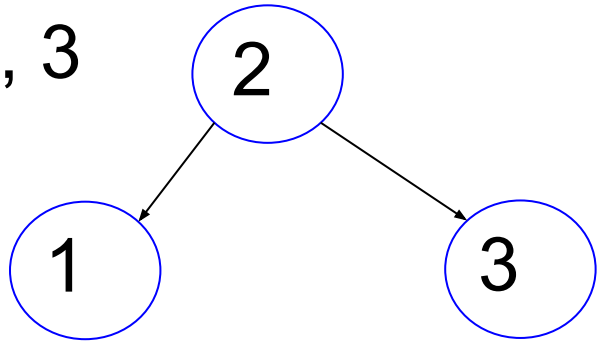
The insert function



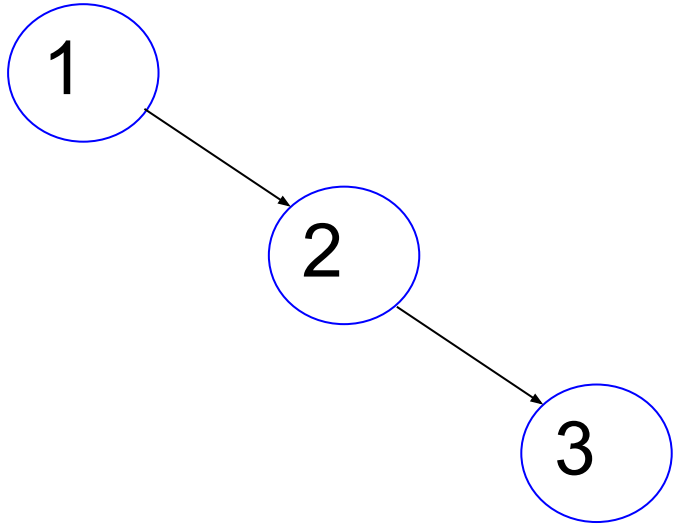
yunglu@purdue.edu

Order of insertion may change tree

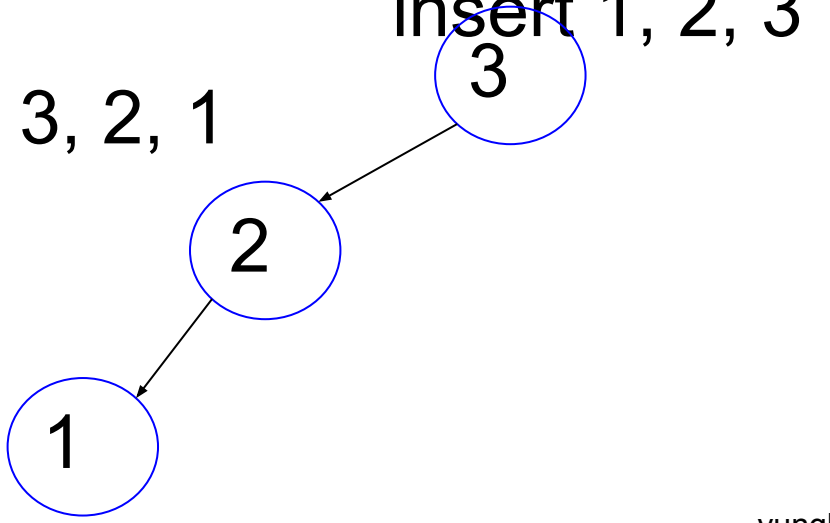
insert 2, 3, 1 or 2, 1, 3



insert 1, 2, 3



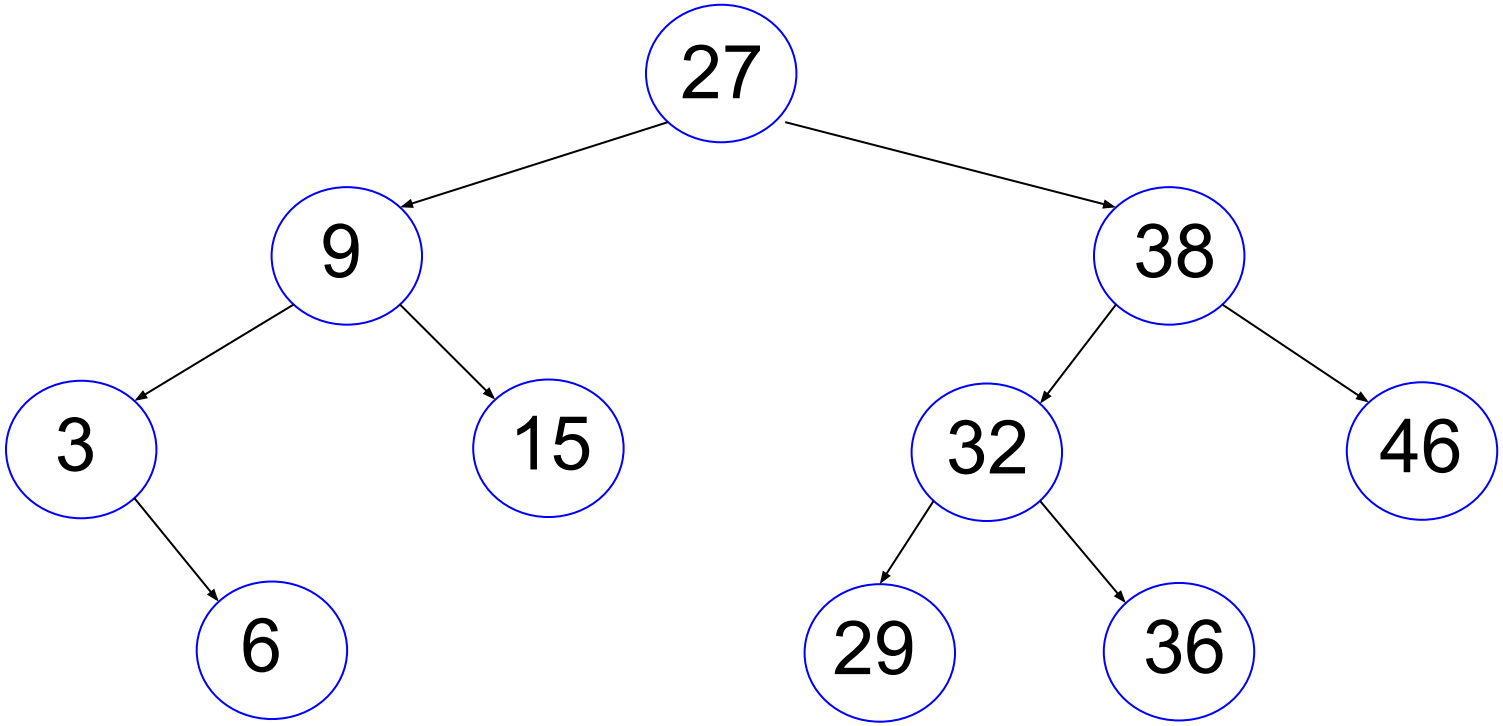
insert 3, 2, 1



yunglu@purdue.edu

Pre-Order

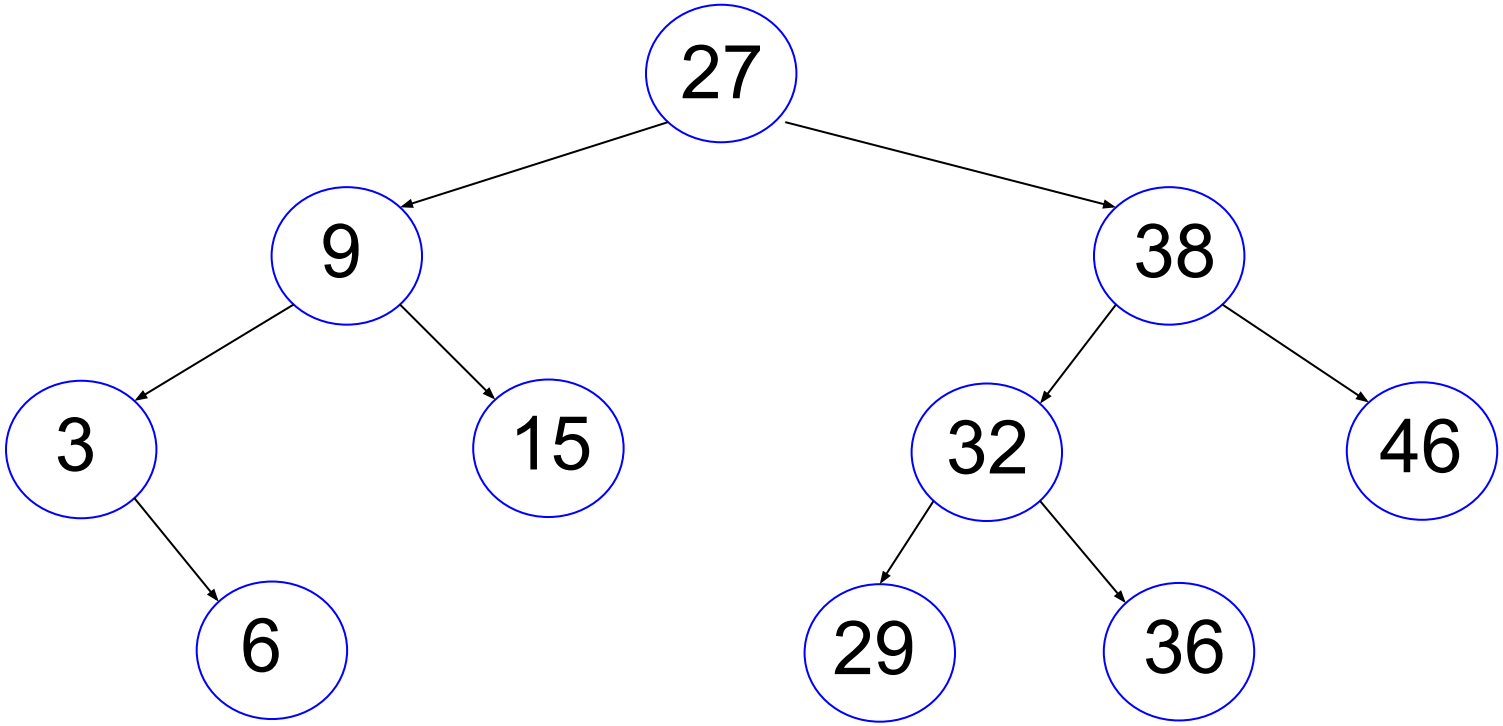
27	9	3	6	15	38	32	29	36	46



yunglu@purdue.edu

In-Order

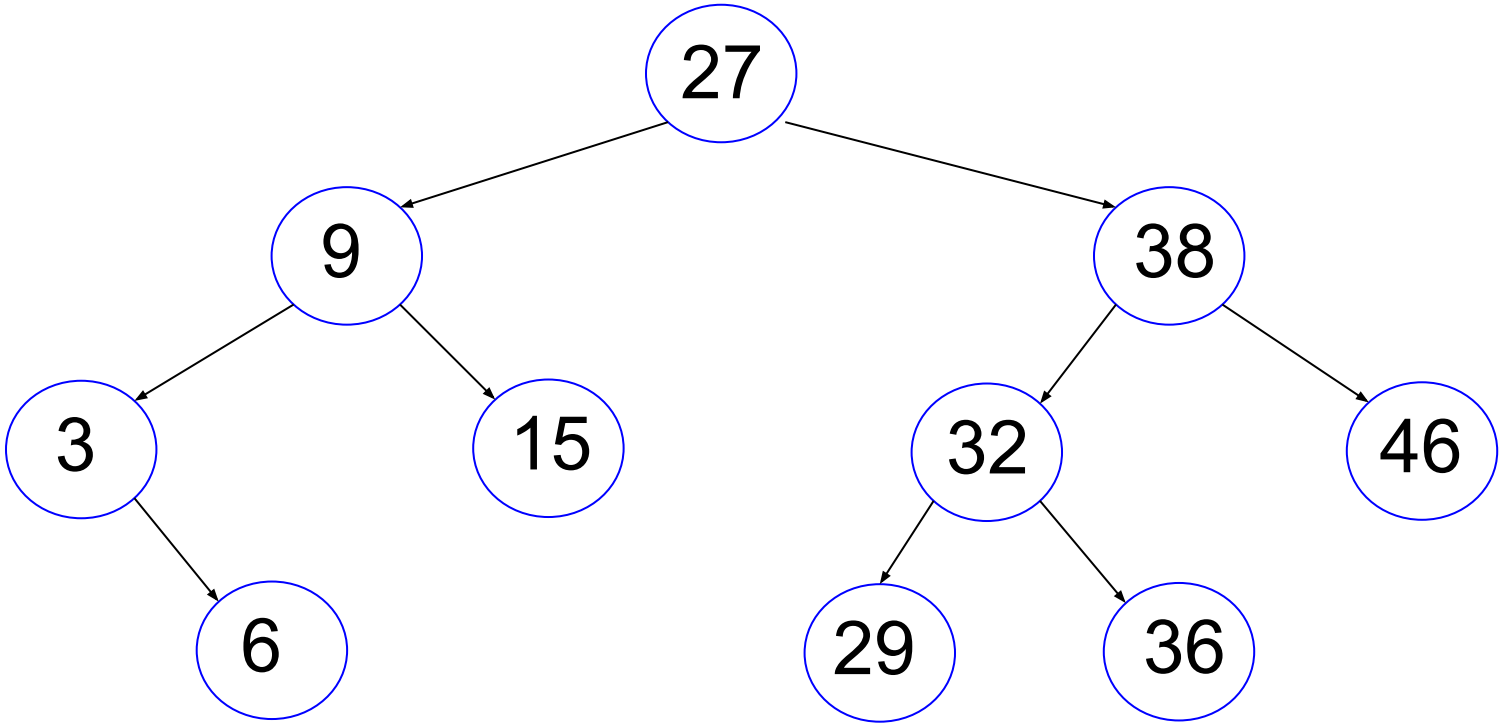
3	6	9	15	27	29	32	36	38	46



yunglu@purdue.edu

Post-Order

6	3	15	9	29	36	32	46	38	27



yunglu@purdue.edu

Binary Trees

- Given 2-traversals of a tree. Can you construct the binary tree?
  - In-order and pre-order
  - In-order and post order
  - pre-order and post-order

- How about 1-traversal?

Homework 15

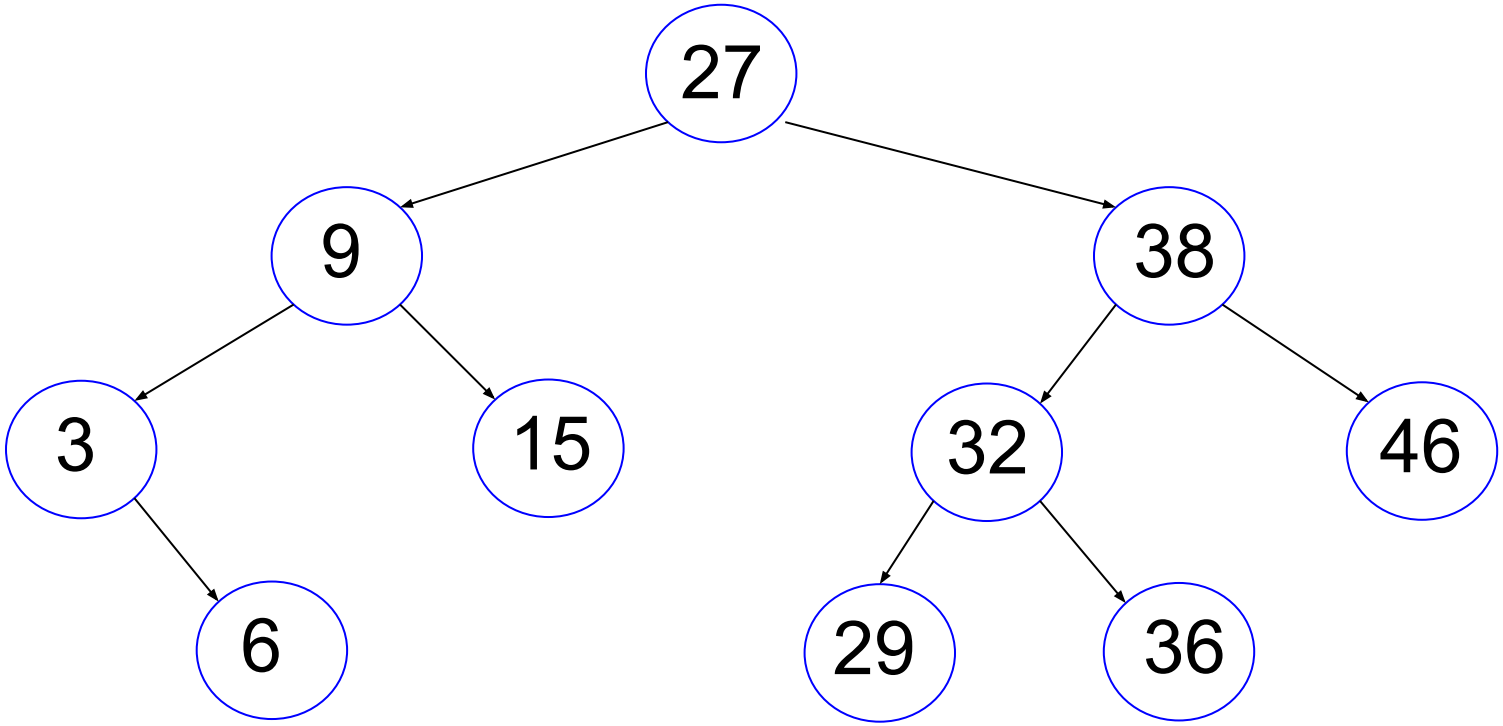
Build Binary Tree from In-Order and Post-Order

Not limited to Binary Search Tree

yunglu@purdue.edu

In-Order	3	6	9	15	27	29	32	36	38	46

Post-Order	6	3	15	9	29	36	32	46	38	27



yunglu@purdue.edu

From in-order and post-order

in-order	3	6	9	15	27	29	32	36	38	46
post-order	6	3	15	9	29	36	32	46	38	27

Root

yunglu@purdue.edu

From in-order and post-order

in-order	3	6	9	15	27	29	32	36	38	46
post-order	6	3	15	9	29	36	32	46	38	27

Root

27

in-order	3	6	9	15	27	29	32	36	38	46
post-order	6	3	15	9	29	36	32	46	38	27

yunglu@purdue.edu

From in-order and post-order

in-order	3	6	9	15	27	29	32	36	38	46
post-order	6	3	15	9	29	36	32	46	38	27

Root

27

in-order	3	6	9	15	27	29	32	36	38	46
post-order	6	3	15	9	29	36	32	46	38	27

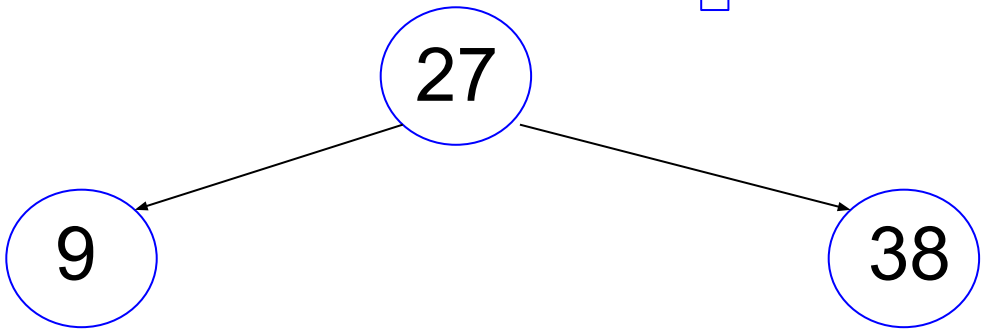
	left subtree of 27					right subtree of 27				
in-order	3	6	9	15	27	29	32	36	38	46
post-order	6	3	15	9	29	36	32	46	38	27

yunglu@purdue.edu

	left subtree of 27					right subtree of 27				
in-order	3	6	9	15	27	29	32	36	38	46
post-order	6	3	15	9	29	36	32	46	38	27

Root

Root



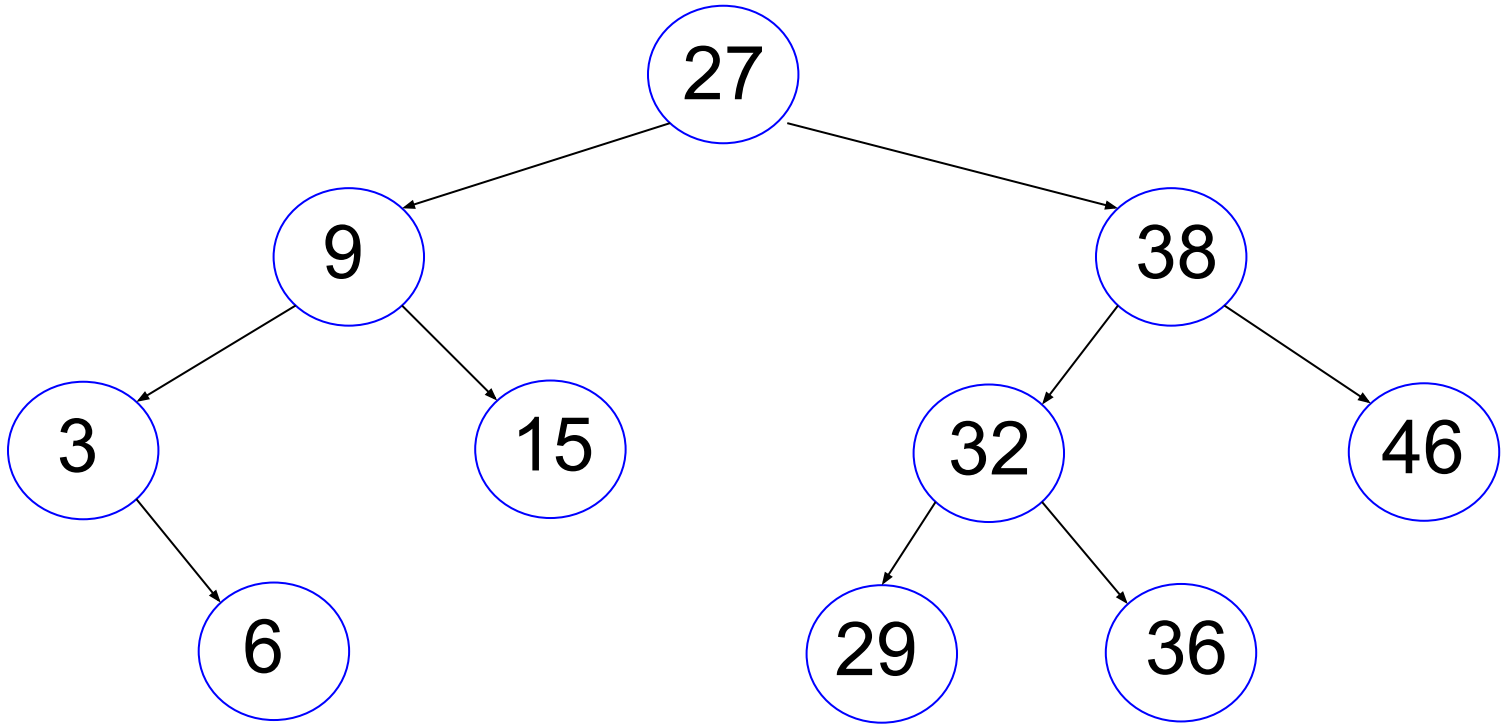
	left subtree of 27					right subtree of 27				
	left of 9					left of 38				
in-order	3	6	9	15	27	29	32	36	38	46
post-order	6	3	15	9	29	36	32	46	38	27

yunglu@purdue.edu

	left subtree of 27					right subtree of 27				
	left of 9					left of 38				
in-order	3	6	9	15	27	29	32	36	38	46
post-order	6	3	15	9	29	36	32	46	38	27

Root

Root



yunglu@purdue.edu

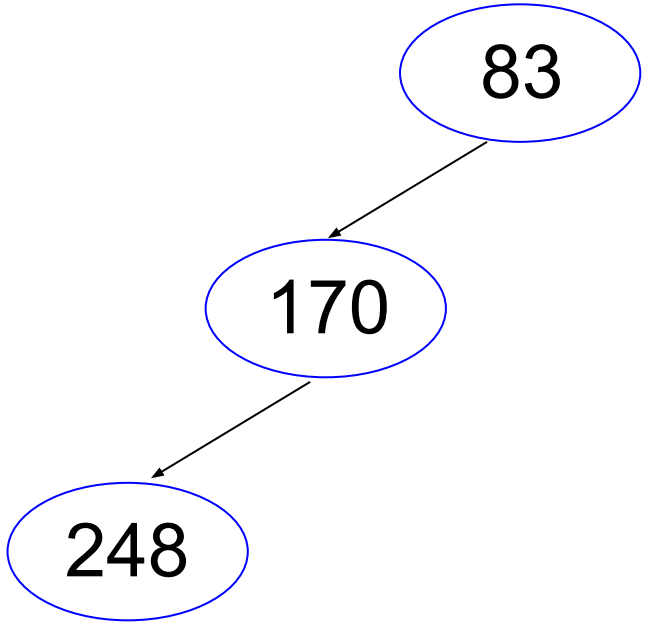
Homework 16

Print the Path from A Node to Root

yunglu@purdue.edu

in-order	248	170	83
post-order	248	170	83

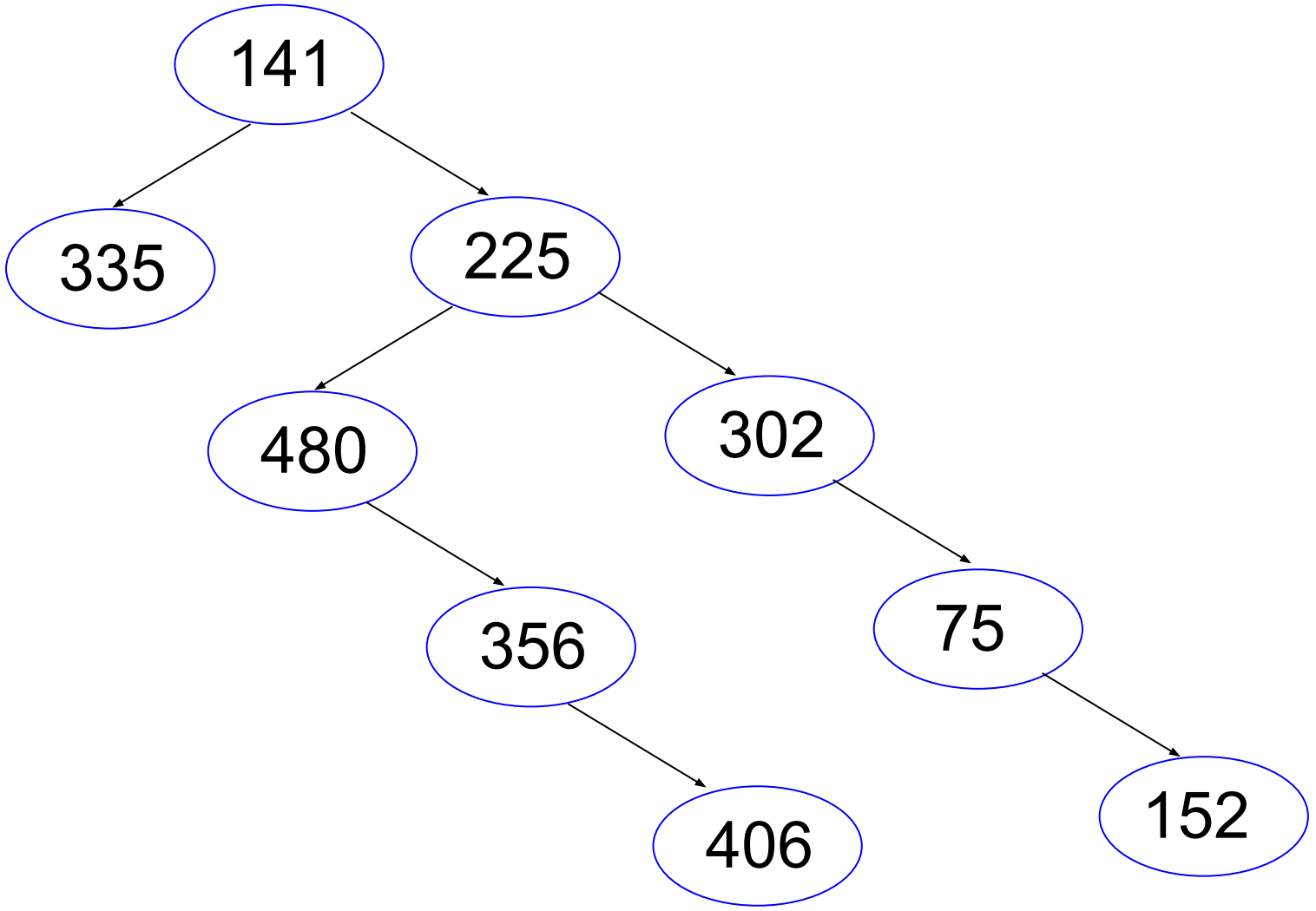
How to get to 248?
From Bottom to
Top
248, 170, 83



yunglu@purdue.edu

in-order	335	141	480	356	406	225	302	75	152
post-order	335	406	356	480	152	75	302	225	141

How to get to 335?
From Bottom to Top
335, 141



yunglu@purdue.edu

Huffman Compression 02

File Format and Reconstructing the Code Tree

yunglu@purdue.edu

Huffman Coding (Compression)

Lossless compression

1. Count the occurrences of the characters (may include symbols and unprintable characters)
2. Sort the characters by their occurrences in the ascending order
3. Take the two least occurrences, make them left and right children of the same parent node, add the occurrences and sort in the ascending order again
4. Continue 3 until only one node is left

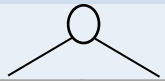
yunglu@purdue.edu

occurrence	4	18	7	22	10	35
letter	#	A	G	c	m	s

⇩ Sort by the occurrences in ascending order

4	7	10	18	22	35
#	G	m	A	c	s

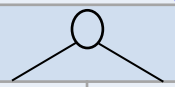
⇩ Make the first two siblings of a binary tree



4	7	10	18	22	35
#	G	m	A	c	s

⇩ The occurrence of the parent is the sum

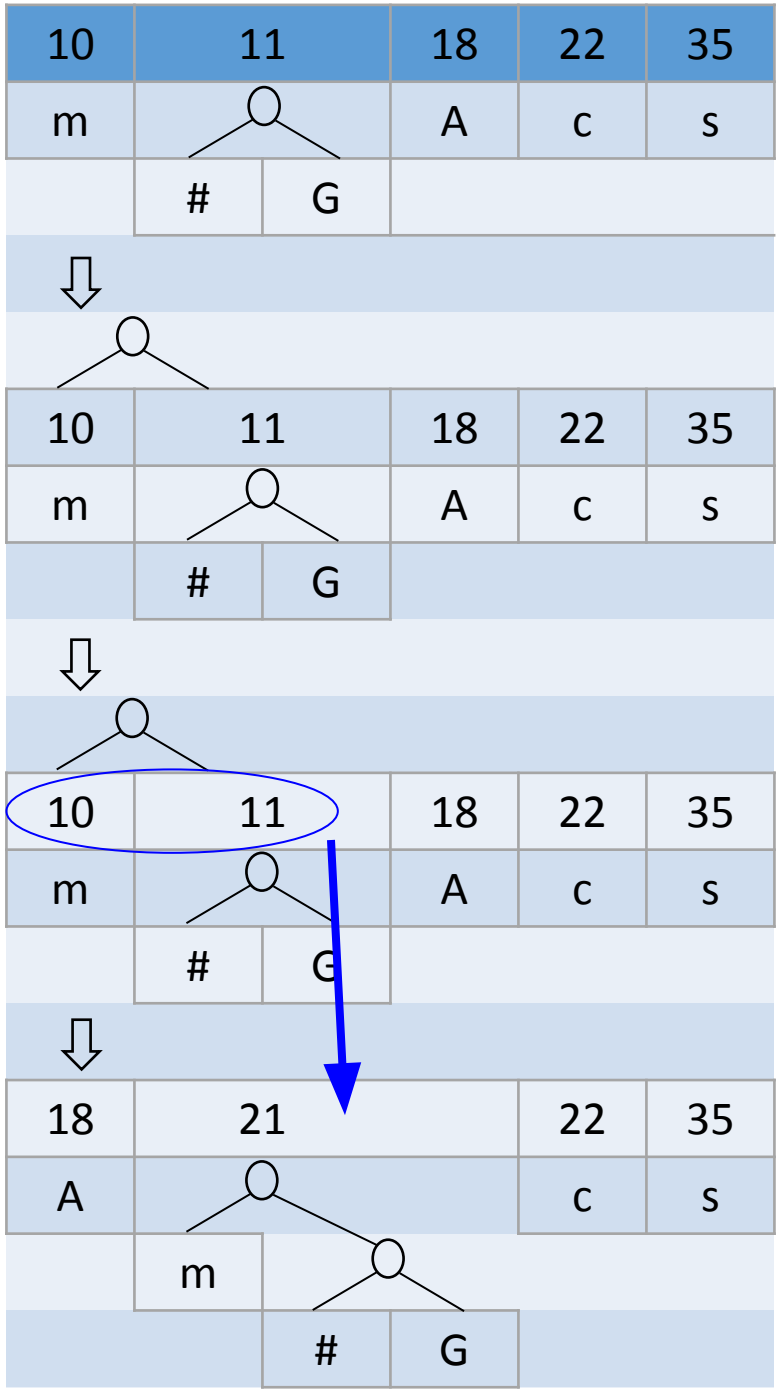
11					

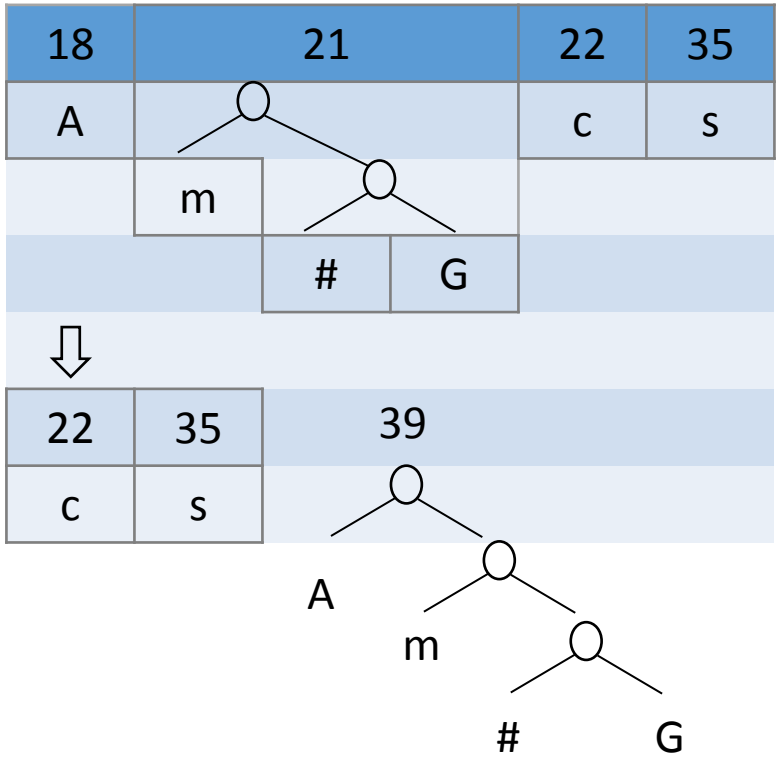


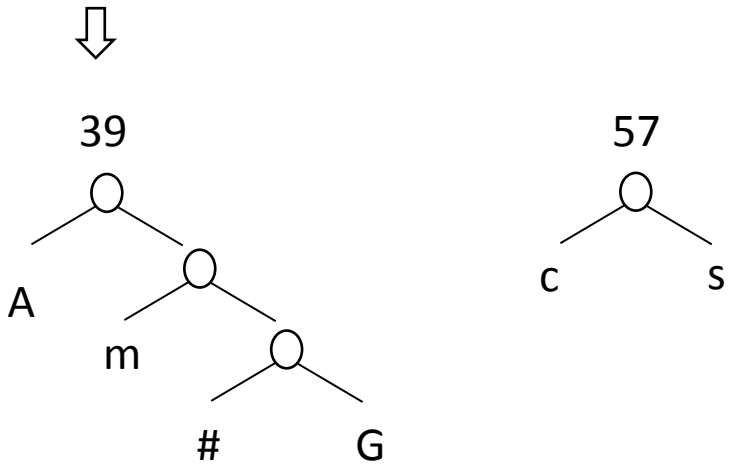
4	7	10	18	22	35
#	G	m	A	c	s

⇩ Insert the parent back in ascending order

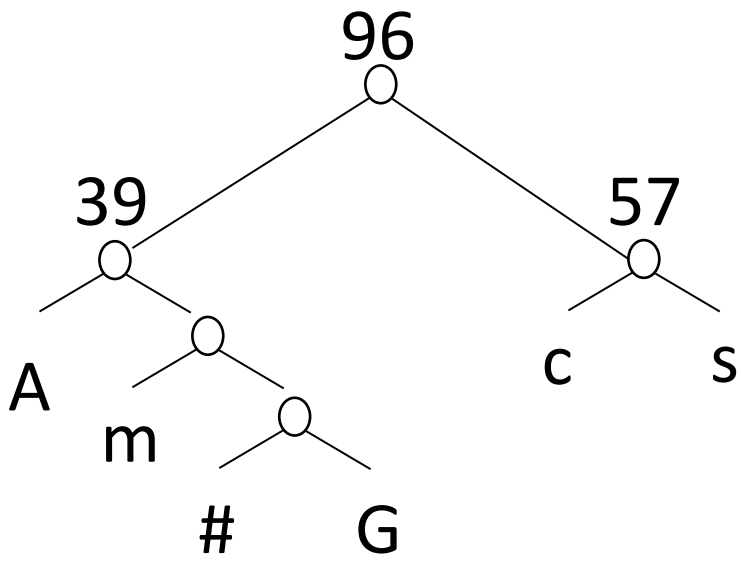
10	11		18	22	35
m			A	c	s
	#	G			

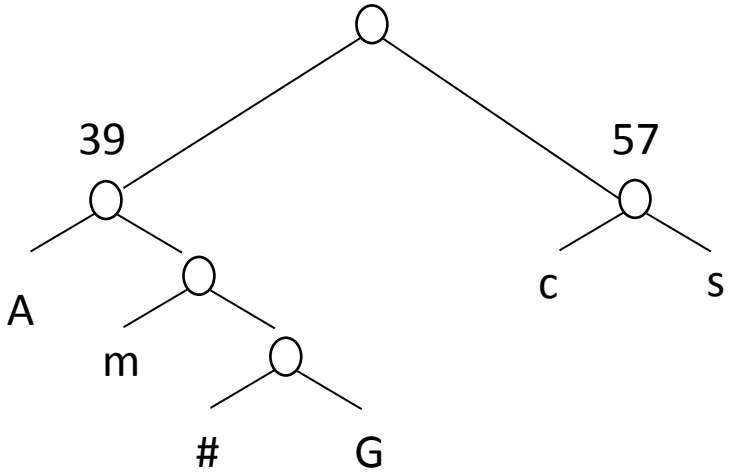

10
11
18
22
35
m
A
c
s
#
G
10
11
18
22
35
m
A
c
s
#
G
10
11
18
22
35
m
A
c
s
#
G
18
21
22
35
A
c
s
m
#
G

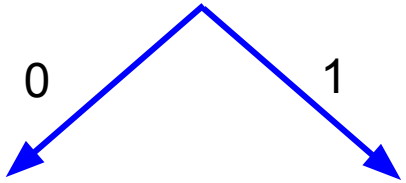




Only the leaf nodes contain characters







character	occurrence	code				length
A	18	0	0			2
m	10	0	1	0		3
#	4	0	1	1	0	4
G	7	0	1	1	1	4
c	22	1	0			2
s	35	1	1			2

yunglu@purdue.edu

character	occurrence	code				length
A	18	0	0			2
m	10	0	1	0		3
#	4	0	1	1	0	4
G	7	0	1	1	1	4
c	22	1	0			2
s	35	1	1			2

- If occurrence (X) < occurrence (Y)

  ⇒ code length (X) ≥ code length (Y)

- code length (X) > code length (Y)

  ⇒ occurrence (X) < occurrence (Y) **WRONG**

yunglu@purdue.edu

character	occurrence	code				length
A	18	0	0			2
m	10	0	1	0		3
#	4	0	1	1	0	4
G	7	0	1	1	1	4
c	22	1	0			2
s	35	1	1			2

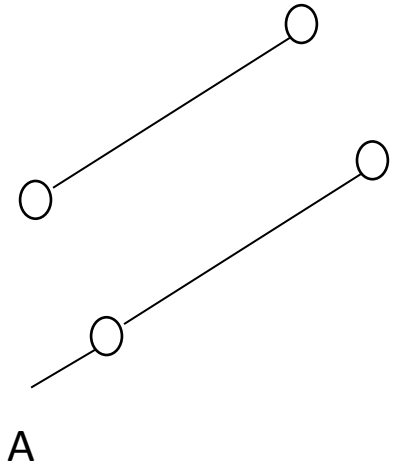
input	A	A	c	s	#	m	G	c	s	A	…
output	0 0	0 0	1 0	1 1	0 1 1 0	0 1 0	0 1 1 1	1 0	1 1	0 0	…

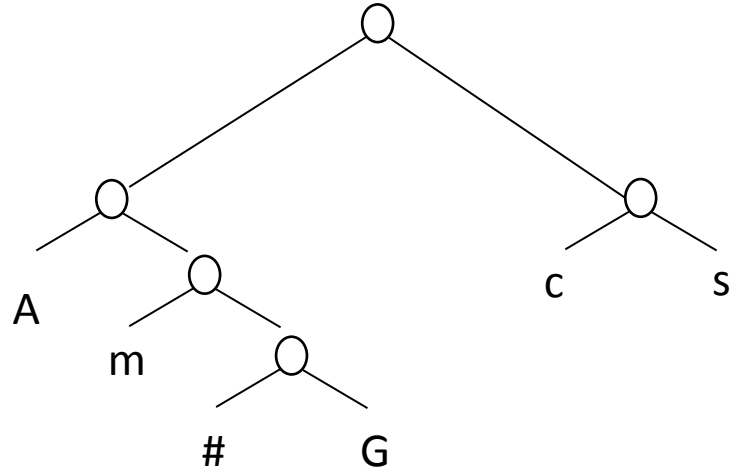
yunglu@purdue.edu

input	0 0 0 0 1 0 1 1 0 1 1 0 0 1 0 0 1 1 1 1 0 1 1 0 0

0

0 0 ⇨ A

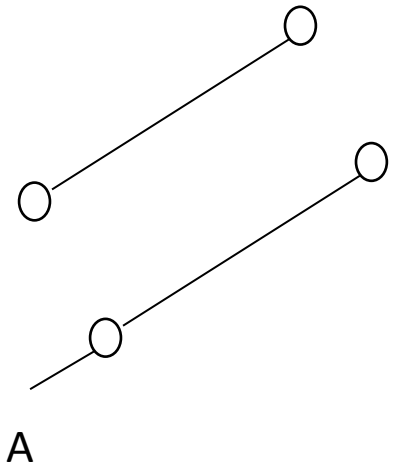




input	0 0	0 0 1 0 1 1 0 1 1 0 0 1 0 0 1 1 1 1 0 1 1 0 0
	A	

0

0 0 A ⇨

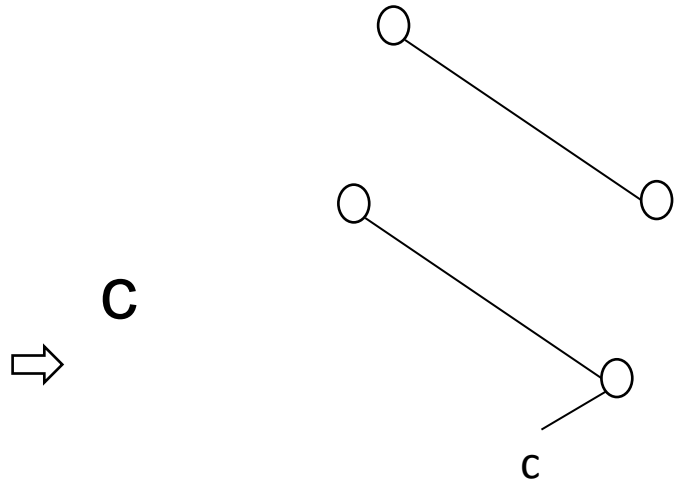


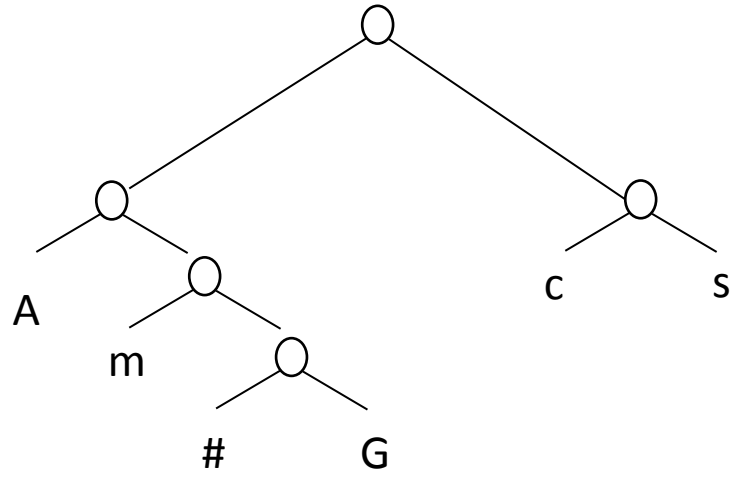
yunglu@purdue.edu

input	0 0	0 0	1 0 1 1 0 1 1 0 0 1 0 0 1 1 1 1 0 1 1 0 0
	A	A	

1

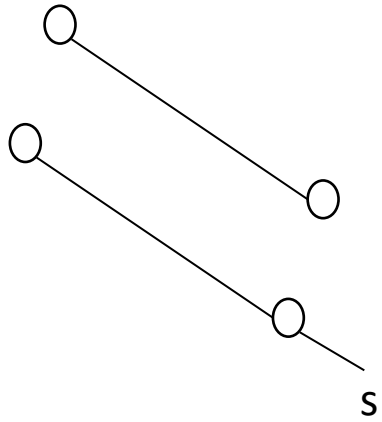
1 0 c





input	0 0	0 0	1 0	1 1 0 1 1 0 0 1 0 0 1 1 1 1 0 1 1 0 0
1	A	A	c	

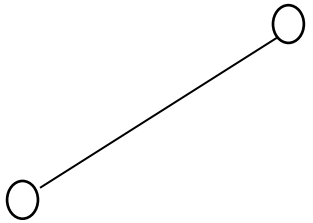
1 1 s



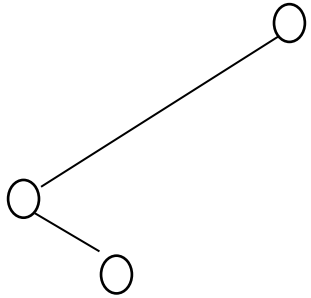
yunglu@purdue.edu

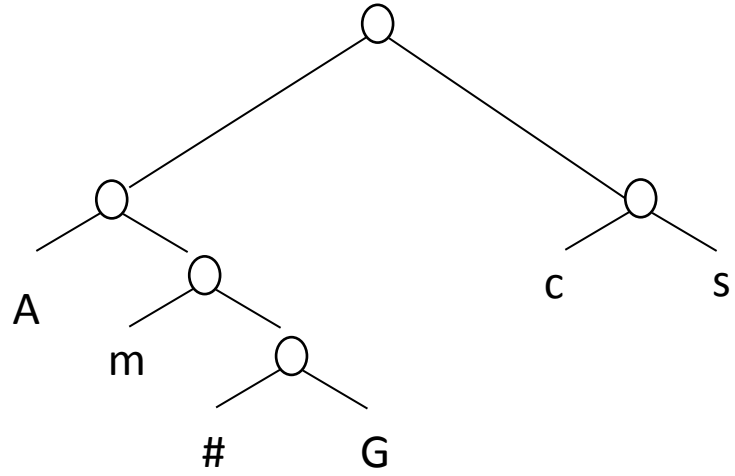
input	0 0	0 0	1 0	1 1	0 1 1 0 0 1 0 0 1 1 1 0 1 1 0 0
	A	A	c	s	

0

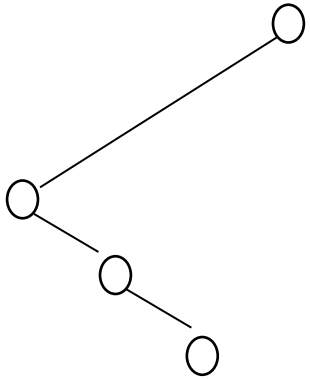


0 1 ⇨



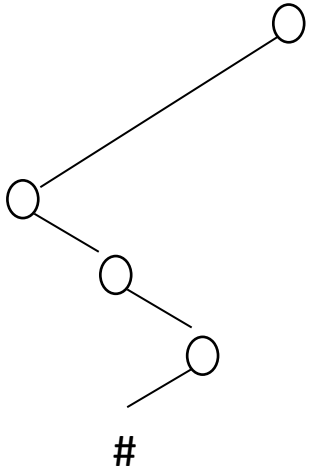


0 1 1



0 1 1 0 #

⇨



yunglu@purdue.edu

input	0 0	0 0	1 0	1 1	0 1 1 0	0 1 0 0 1 1 1 1 0 1 1 0 0
	A	A	c	s	#	

- 0 goes to the left
- 1 goes to the right
- If reach a leaf node, output the character
- go back to the root

- Characters are stored in only leaf nodes (by construction)

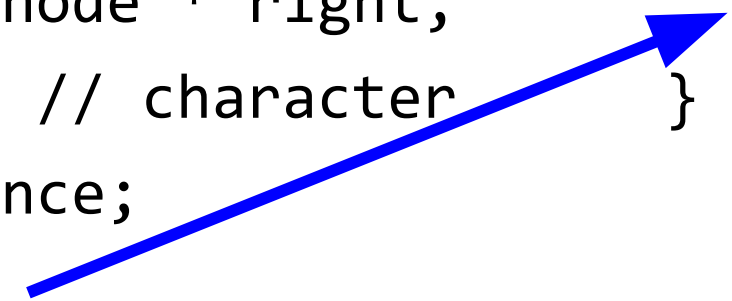
yunglu@purdue.edu

How to build the compression tree

Ch 24 in https://github.com/yunghsianglu/IntermediateCProgramming

```
typedef struct treenode
{
  struct treenode * left;
  struct treenode * right;
  char value; // character
  int occurrence;
} TreeNode;
```

```
typedef struct listnode
{
  struct listnode * next;
  TreeNode * tnptr;
} ListNode;
```



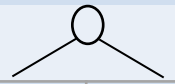
yunglu@purdue.edu

occurrence	4	18	7	22	10	35
letter	#	A	G	c	m	s

⇩ Sort by the occurrences in ascending order

4	7	10	18	22	35
#	G	m	A	c	s

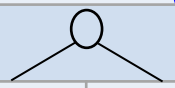
⇩ Make the first two siblings of a binary tree



4	7	10	18	22	35
#	G	m	A	c	s

⇩ The occurrence of the parent is the sum

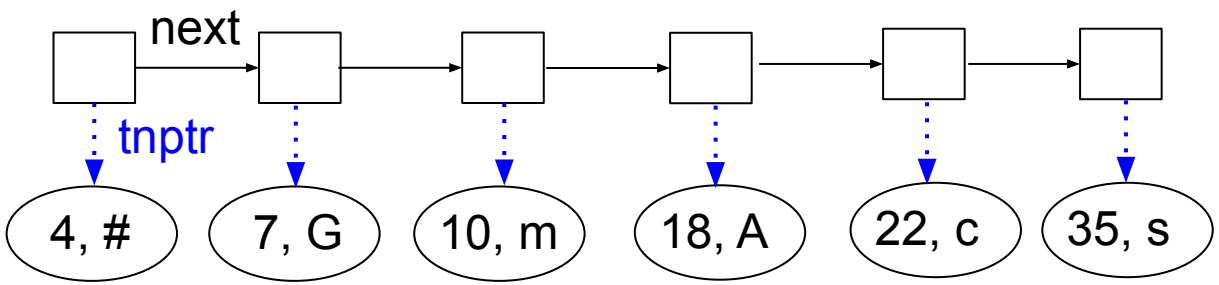
11					

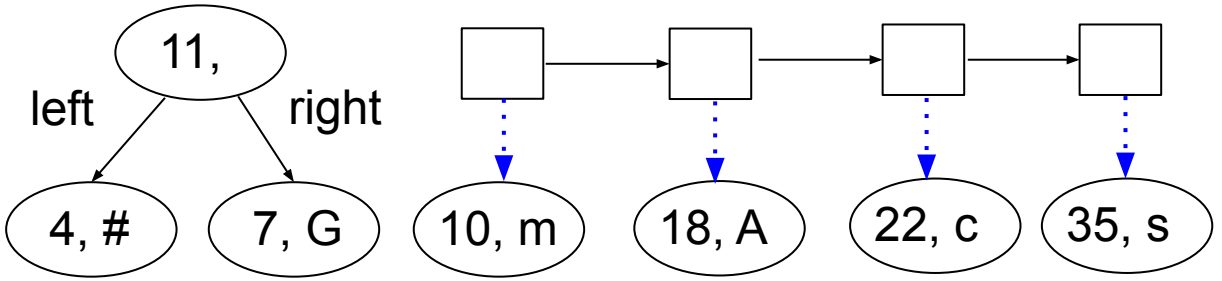


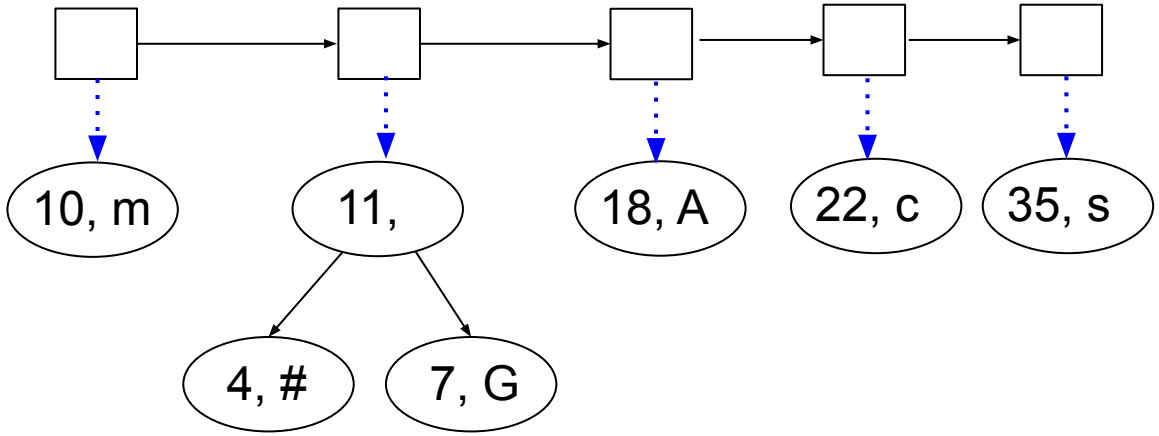
4	7	10	18	22	35
#	G	m	A	c	s

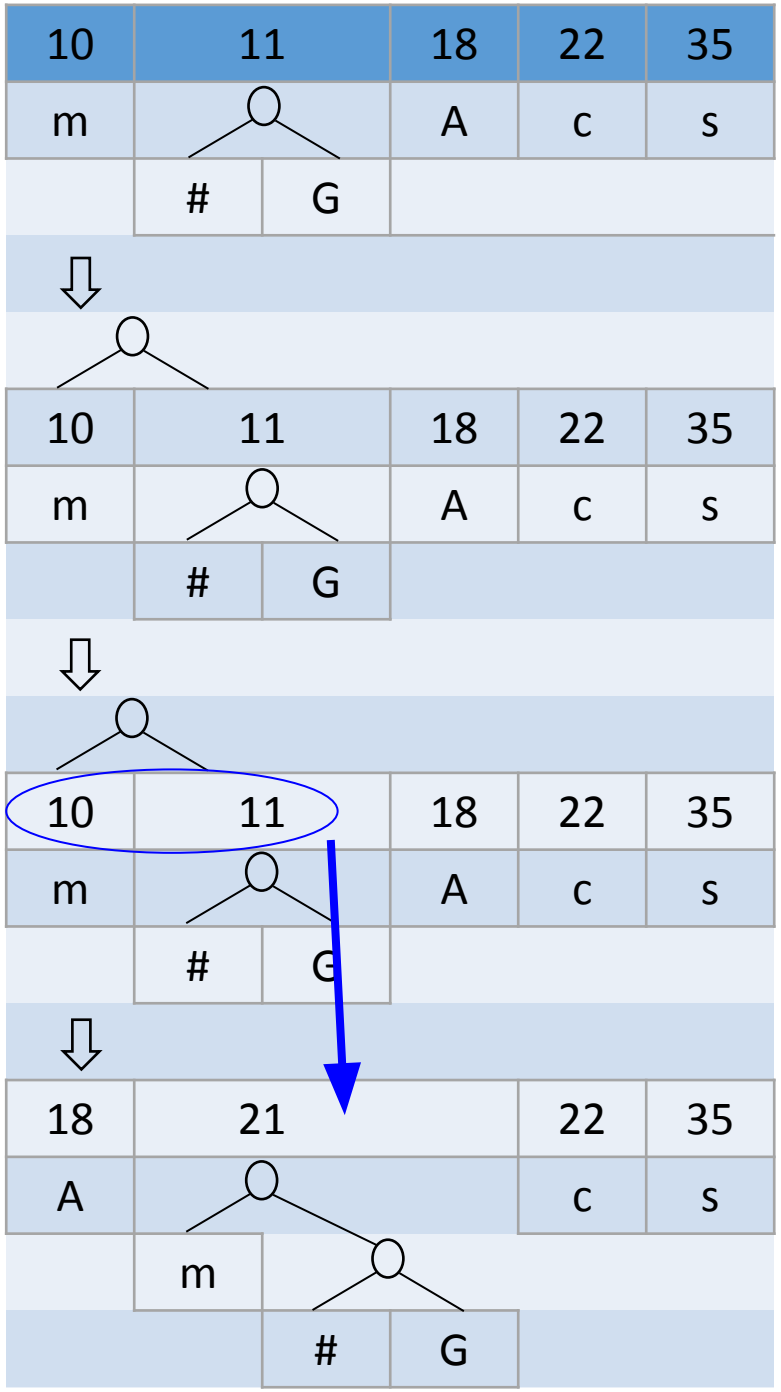
⇩ Insert the parent back in ascending order

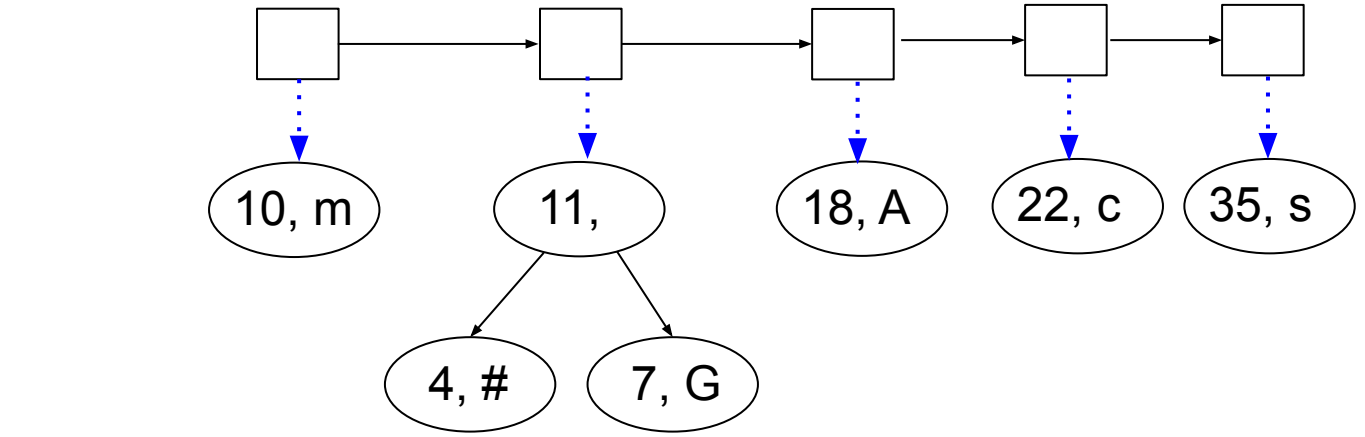
10	11		18	22	35
m			A	c	s
	#	G			

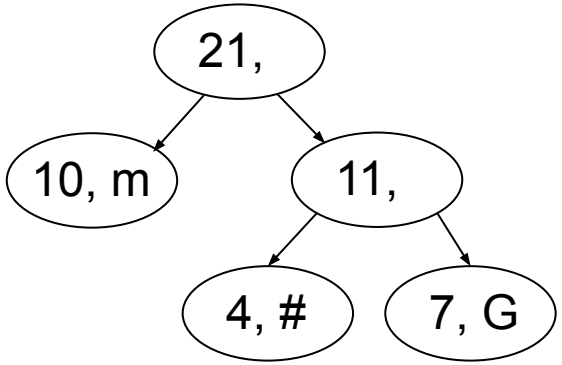


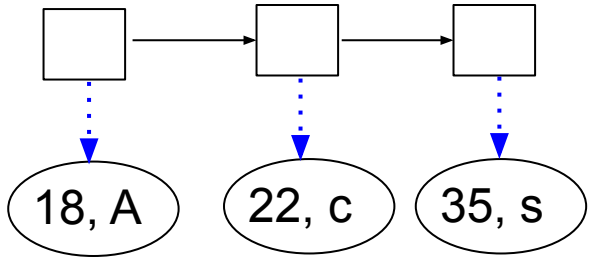


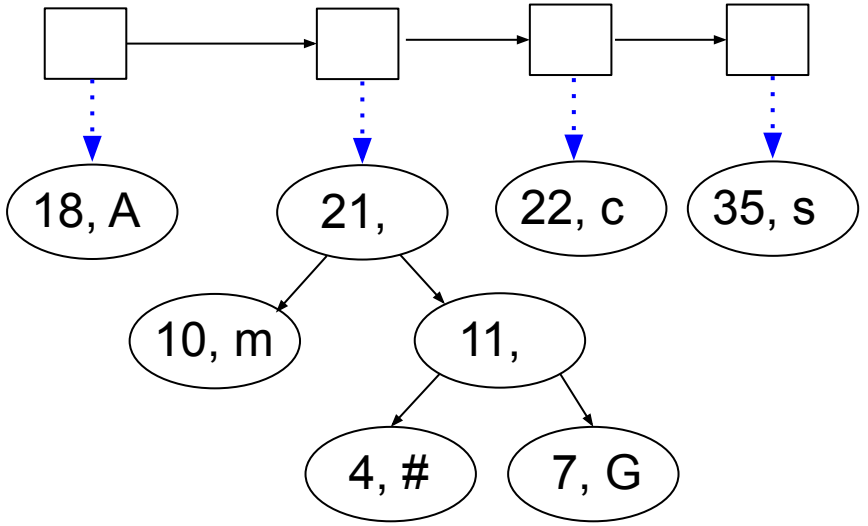



10
11
18
22
35
m
A
c
s
#
G
10
11
18
22
35
m
A
c
s
#
G
10
11
18
22
35
m
A
c
s
#
G
18
21
22
35
A
c
s
m
#
G


10, m
11,
18, A
22, c
35, s
4, #
7, G

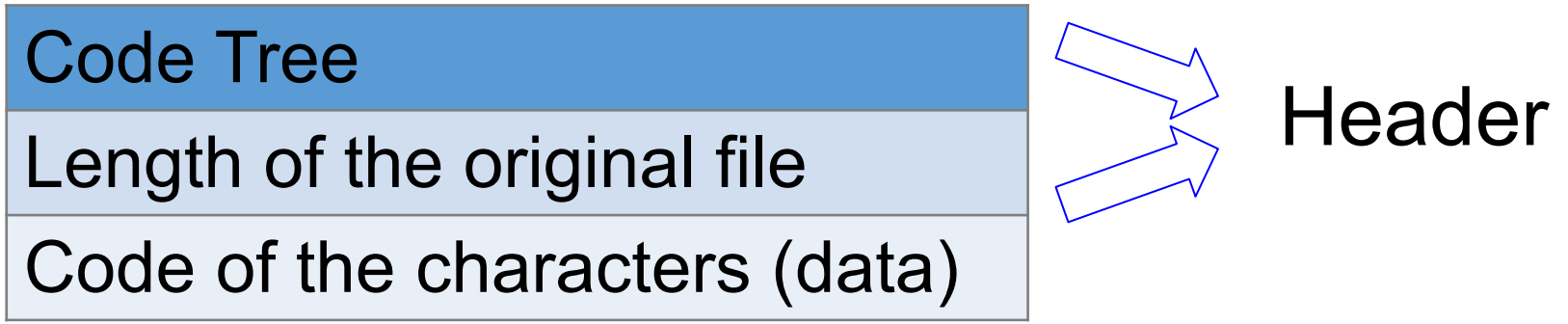

21,
10, m
11,
4, #
7, G


18, A
22, c
35, s


18, A
21,
22, c
35, s
10, m
11,
4, #
7, G

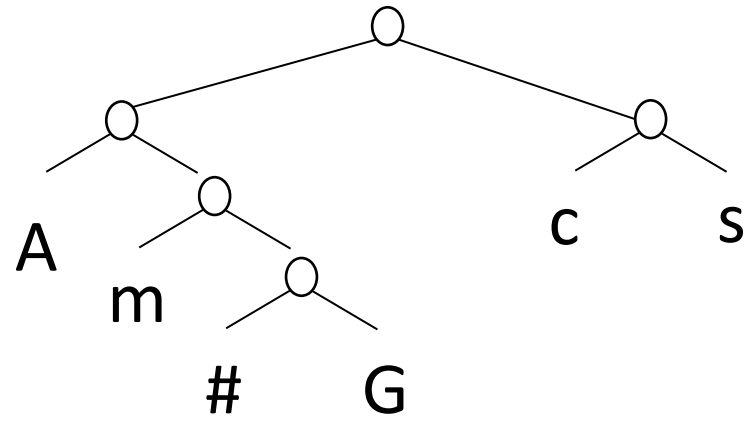
Format of File using Huffman Codes

- The code depends on the data.
- Two different files may use different sets of characters.
- The same characters may have different occurrences.
- Different formats can be used as long as the compression and decompression programs agree



yunglu@purdue.edu

Express Code Table

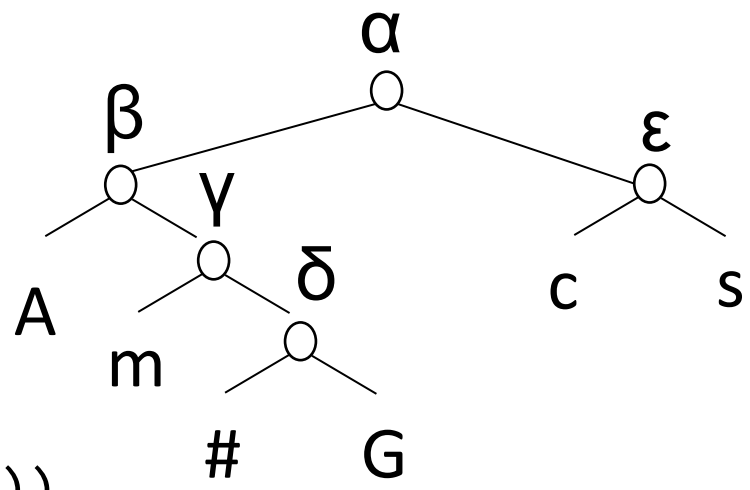


- Need a way to uniquely describe the tree
- This class use “post-order” traversal:
  - If it is a leaf node, print 1 followed by the character
  - If it is a non-leaf node, print 0

```
void Tree_printPostorder(TreeNode *tn)
{
  if (tn == NULL) { return; }
  Tree_printPostorder(tn -> left);
  Tree_printPostorder(tn -> right);
  if ((tn -> left) == NULL) && (tn -> right) == NULL)) // leaf node
        { printf(“1%c”, tn -> value);  }
  else { printf(“0”); }
}
```

yunglu@purdue.edu

```
void Tree_printPostorder(TreeNode *tn)
{
  if (tn == NULL) { return; }
  Tree_printPostorder(tn -> left);
  Tree_printPostorder(tn -> right);
  if ((tn -> left) == NULL) && (tn -> right) == NULL))
  { printf("1%c", tn -> value);  }
  else { printf("0"); }
}
```



left subtree of α					right subtree of α			
left subtree of β	right subtree of β			β	left subtree of ε	right subtree of ε	ε	α
A				β				

A	m	#	G	δ	γ	β	c	s	ε	α
1A	1m	1#	1G	0	0	0	1c	1s	0	0

⇒ 1A 1m 1# 1G 0 0 0 1c 1s 0 0

yunglu@purdue.edu

From post-order expression to tree

1A 1m 1# 1G 0 0 0 1c 1s 0 0

- 1 and 0 are “control”
- the characters are “data”
- control 1: create a tree node of character and add to list
- control 0: take the most recent two tree nodes and make them siblings, add the parent node back

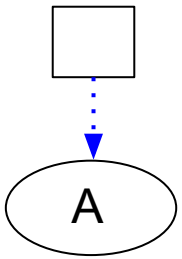
yunglu@purdue.edu

1A 1m 1# 1G 0 0 0 1c 1s 0 0

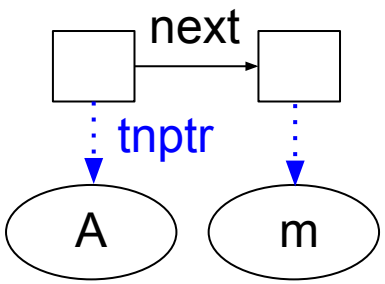
list node tree node

- 1: create a tree node and add to list
- 0: take the most recent two tree nodes and make them siblings

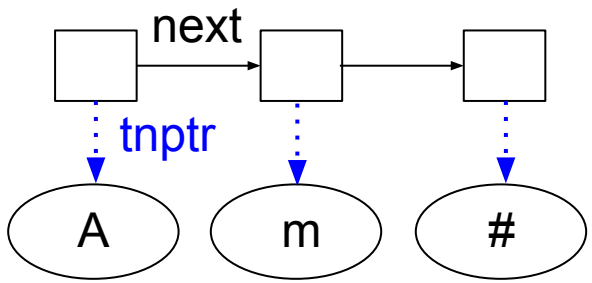
1A



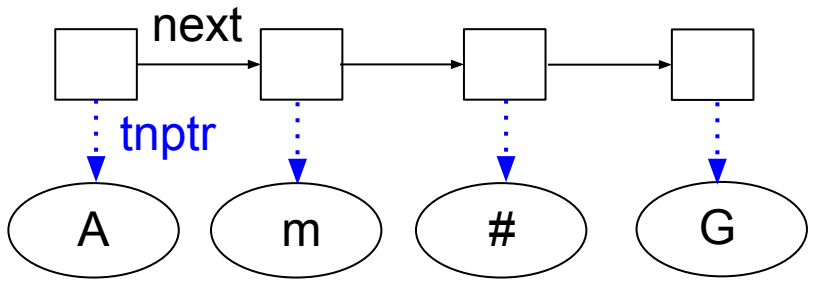
1A 1m



1A 1m 1#



1A 1m 1# 1 G

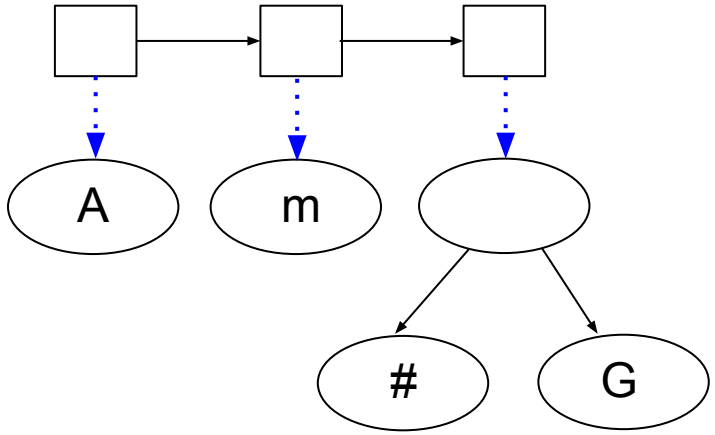


yunglu@purdue.edu

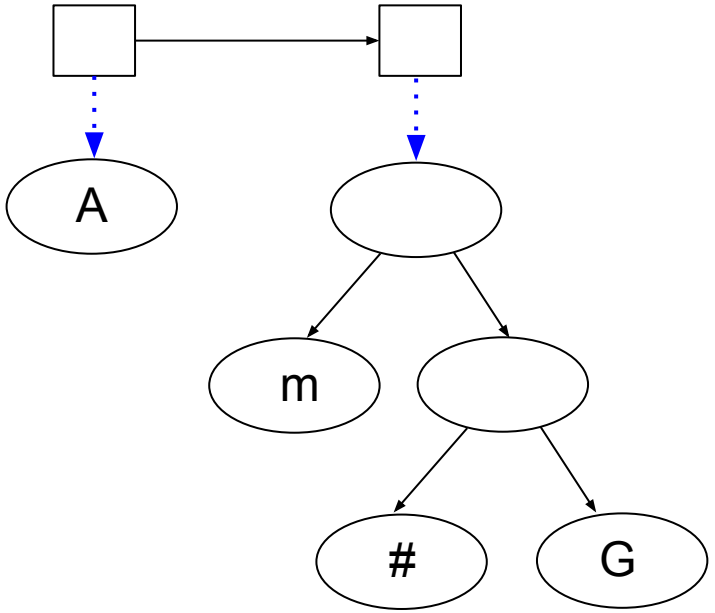
1A 1m 1# 1G 0 0 0 1c 1s 0 0

- 1: create a tree node and add to list
- 0: take the most recent two tree nodes and make them siblings

1A 1m 1# 1 G 0



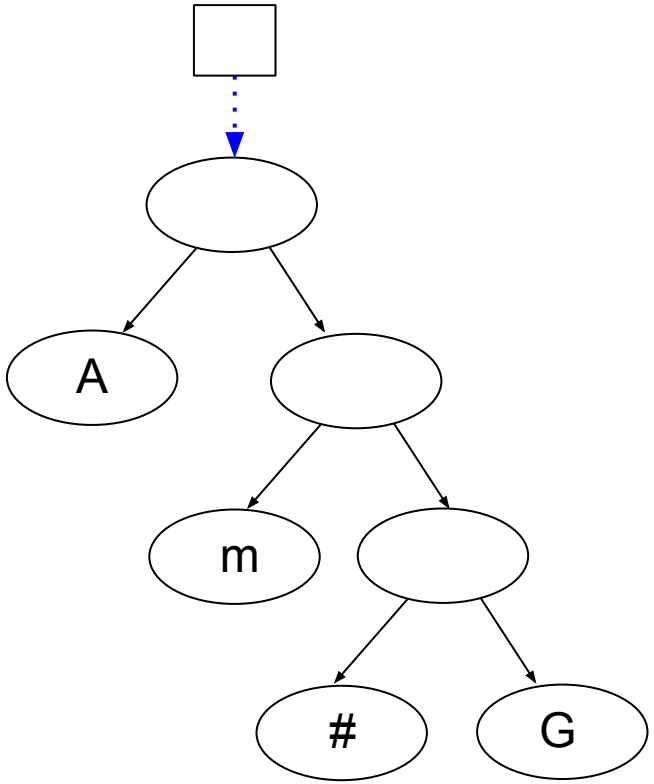
1A 1m 1# 1 G 0 0



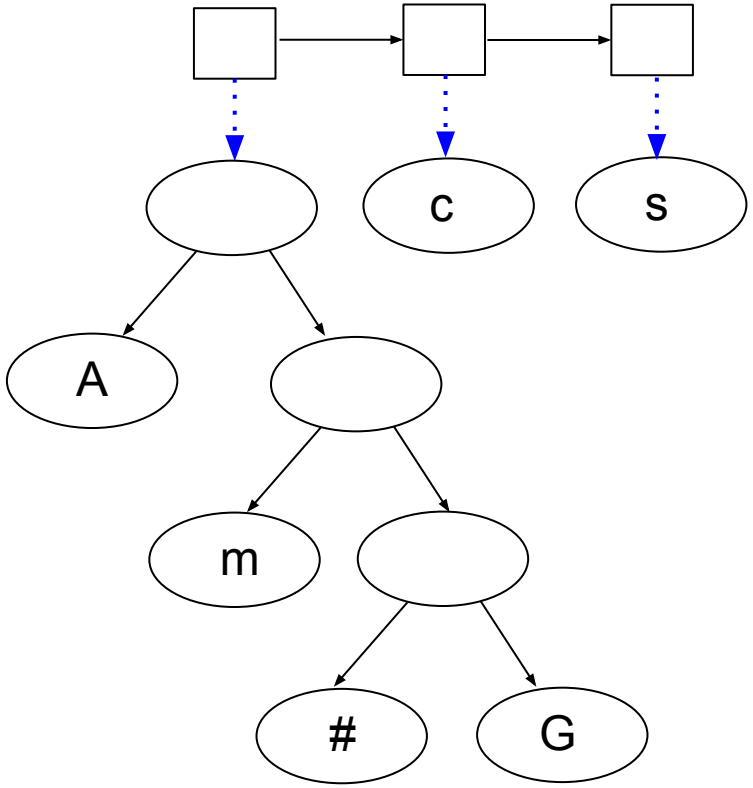
yunglu@purdue.edu

1A 1m 1# 1G 0 0 0 1c 1s 0 0

1A 1m 1# 1 G 0 0 0



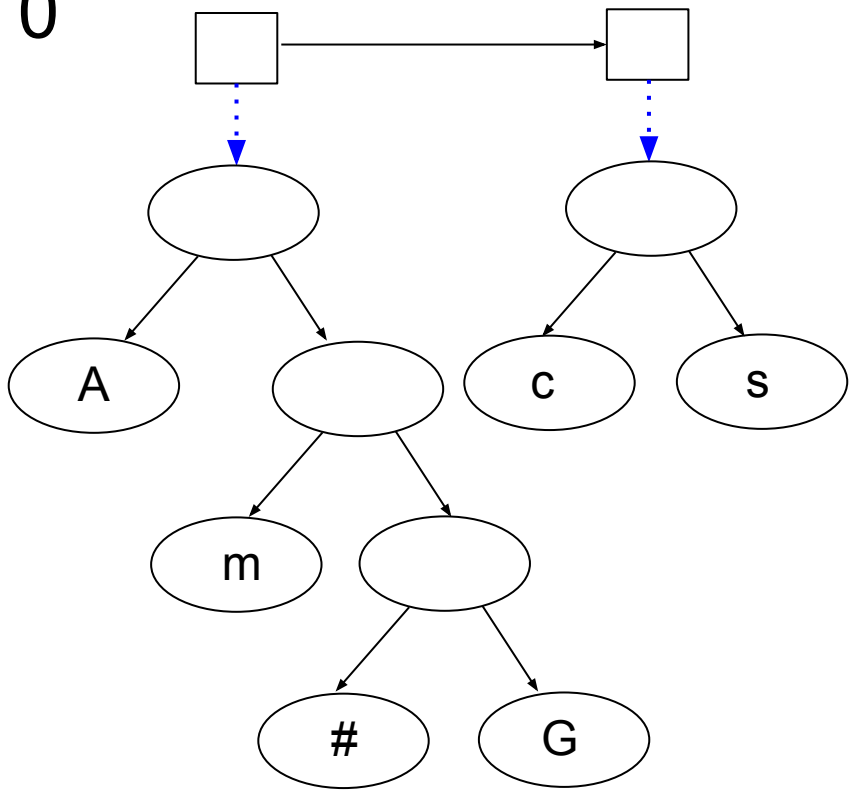
1A 1m 1# 1 G 0 0 0 1c 1s



yunglu@purdue.edu

1A 1m 1# 1G 0 0 0 1c 1s 0 0

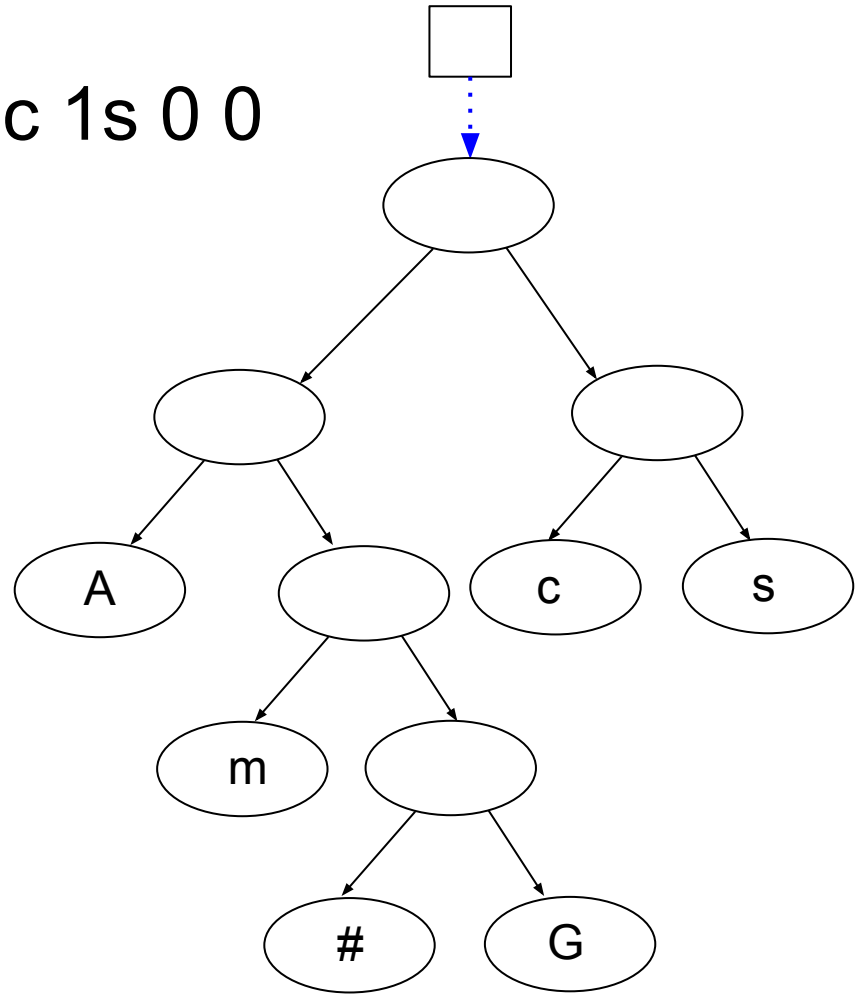
1A 1m 1# 1 G 0 00 1c 1s 0

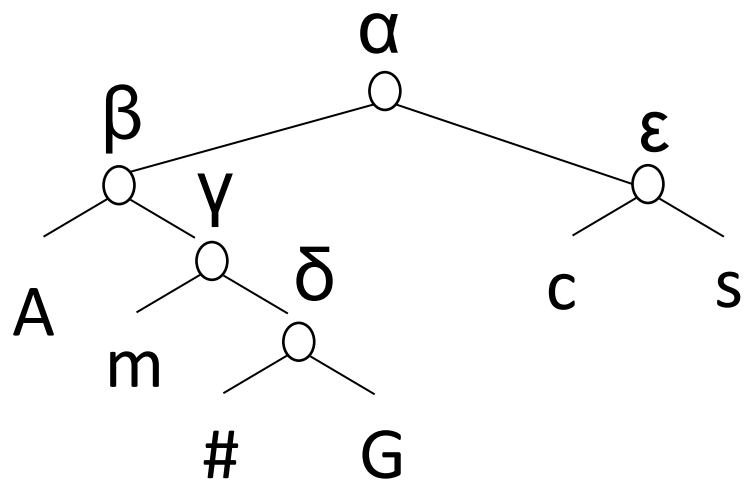


yunglu@purdue.edu

1A 1m 1# 1G 0 0 0 1c 1s 0 0

1A 1m 1# 1 G 0 00 1c 1s 0 0



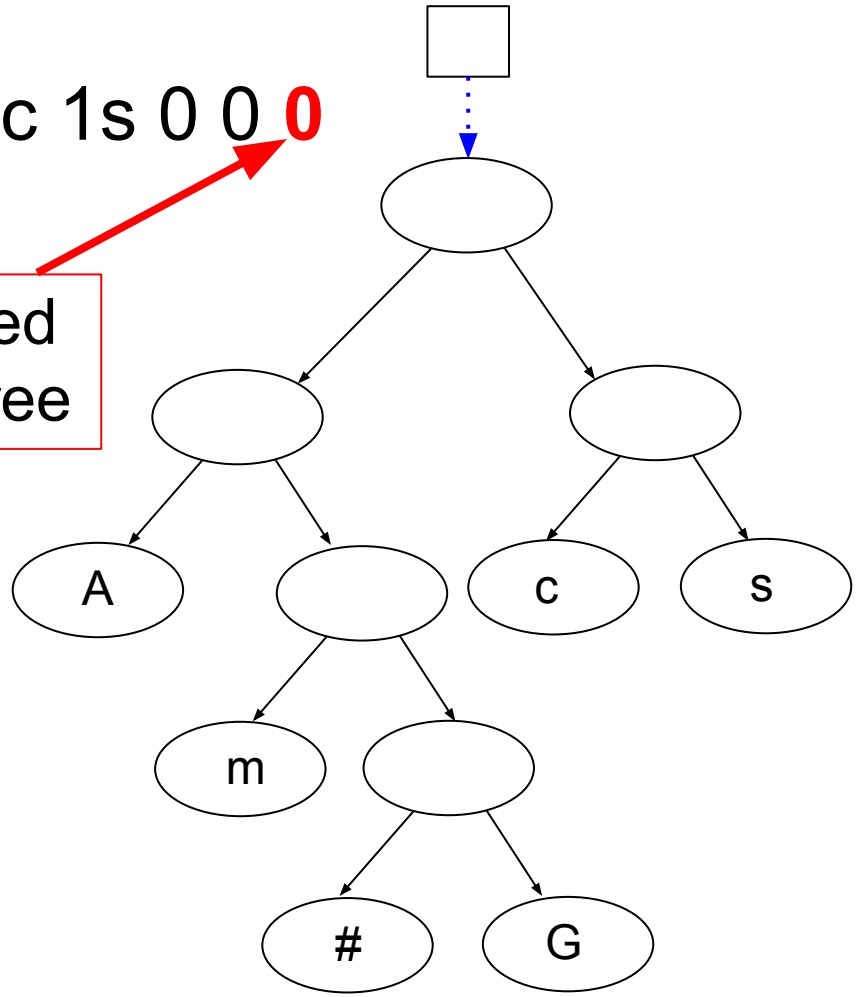


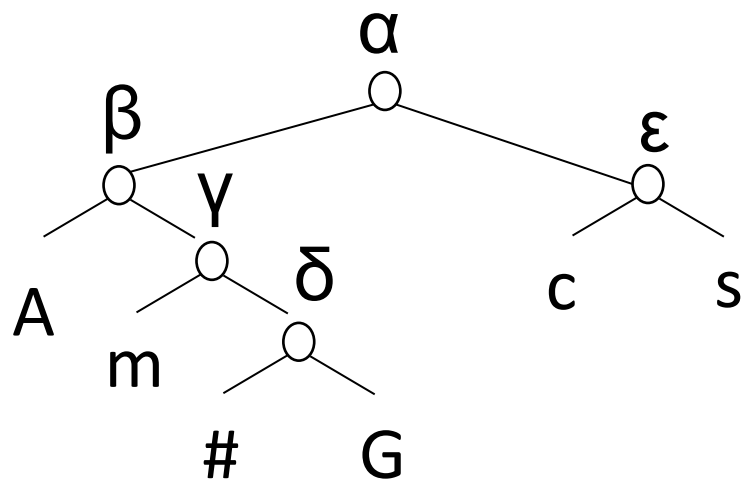
yunglu@purdue.edu

1A 1m 1# 1G 0 0 0 1c 1s 0 0

1A 1m 1# 1 G 0 00 1c 1s 0 0 **0**

One additional zero is added to indicate the end of the tree





yunglu@purdue.edu

Frequently Asked Questions

Q: 1 and 0 are used for control. Does that mean this method cannot encode 1 or 0?

Q: Does the tree need to specify the occurrence?

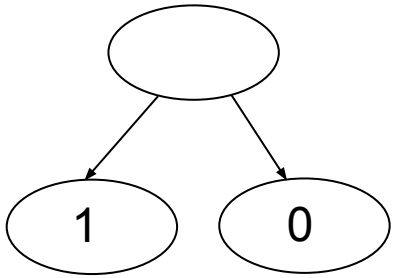
yunglu@purdue.edu

Frequently Asked Questions

Q: 1 and 0 are used for control. Does that mean this method cannot encode 1 or 0?

A: Yes. The method can handle 1 and 0 as data. For example

11 10 0 creates



Q: Does the tree need to specify the occurrence?

A: No need.

yunglu@purdue.edu

Q: Will this method create a unique tree?

Q: How many 1 and 0 are needed?

Q: What is the simplest tree?

Q: What is the simplest tree with something?

Q: How is the length expressed?
A: 32-bit integer

Code Tree
Length of the original file
Code of the characters (data)

yunglu@purdue.edu

Q: Will this method create a unique tree?

A: Yes.

Q: How many 1 and 0 are needed?

A: If there are n leaf nodes, there are n ones for control and n – 1 zeros for non-leaf nodes. Adding the final 0, there are n zeros.

Q: What is the simplest tree?

A: The simplest tree has nothing.

Q: What is the simplest tree with something?

A: 1 x 0 is a tree with one node only.

Q: How is the length expressed?

A: 32-bit integer

Code Tree
Length of the original file
Code of the characters (data)

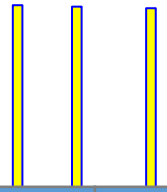
yunglu@purdue.edu

Huffman Coding Increases File Sizes

- 1A 1m 1# 1 G 0 00 1c 1s 0 0 0
- + 4 bytes length
- +

Code Tree
Length of the original file
Code of the characters (data)

data	A	A	c	s	#	m	G	c	s	A	10 bytes	…
code	0 0	0 0	1 0	1 1	0 1 1 0	0 1 0	0 1 1 1	1 0	1 1	0 0	25 bytes	…



Each should be expressed by one bit, not one byte

data	A	A	c	s	4 bytes
code	0 0	0 0	1 0	1 1	1 byte

One bit can express 0 or 1.
One byte has 8 bits and can express 0 to 255.

yunglu@purdue.edu

C does not have bit type

- Most programming languages do not have “bit” types
- C uses **unsigned char** to store a byte as the smallest unit
- We need to use “bitwise operations” to set, reset, and test bits
- Commonly used operations: AND, OR, SHIFT RIGHT, SHIFT LEFT
- AND (&) OR (|)

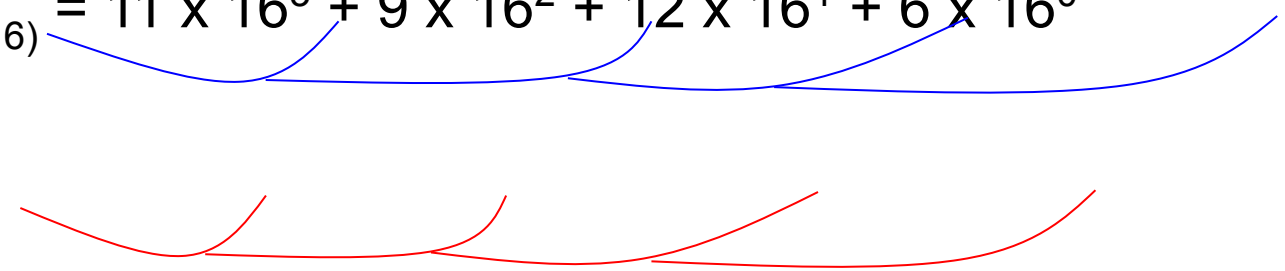
&	1	0
1	1	0
0	0	0

|	1	0
1	1	1
0	1	0

yunglu@purdue.edu

Number Systems (Review)

- Decimal: base 10, 10 digits ⇒ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- Binary: base 2, 2 digits ⇒ 0, 1
- Hexadecimal: base 16, 16 digits ⇒ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
- Octal: base 8, 8 digits ⇒ 0, 1, 2, 3, 4, 5, 6, 7
- $1234_{(10)} = 1 \times 10^3 + 2 \times 10^2 + 3 \times 10^1 + 4 \times 10^0$
- $1011_{(2)} = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$
- $B9C6_{(16)} = 11 \times 16^3 + 9 \times 16^2 + 12 \times 16^1 + 6 \times 16^0$



yunglu@purdue.edu

Review:

- $512.34_{(10)} = 5 \times 10^2 + 1 \times 10^1 + 2 \times 10^0 + 3 \times 10^{-1} + 4 \times 10^{-2}$
- $110.11_{(2)} = 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2}$
- $7B9.C6_{(16)} = 7 \times 16^2 + 11 \times 16^1 + 9 \times 16^0 + 12 \times 16^{-1} + 6 \times 16^{-2}$
- $534_{(10)} = 512 + 22 = 2 \times 16^2 = 1 \times 16^1 + 6 \times 16^0 = 216_{(16)}$
- $16_{(10)} = 2^4 = 10000_{(2)}$
- $D_{(16)} = 13_{(10)} = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 1101_{(2)}$

yunglu@purdue.edu

```
unsigned char a = 0x3D;  // 0011 1101
// 0X means hexadecimal
unsigned char b = 0x96;  // 1001 0110
unsigned char c = a & b; // 0001 0100 = 0X14
unsigned char d = a | b; // 1011 1111 = 0XBF
unsigned char e = 0b10000110; // error, not standard C
```

a	0	0	1	1	1	1	0	1
b	1	0	0	1	0	1	1	0
&	0	0	0	1	0	1	0	0
|	1	0	1	1	1	1	1	1

most significant bit (MSB)

least significant bit (LSB)

yunglu@purdue.edu

```
unsigned char a = 0x3D;  // 0011 1101
unsigned char b = 0x96;  // 1001 0110
unsigned char c = a & b; // 0001 0100 = 0X14
unsigned char d = a | b; // 1011 1111 = 0XBF
unsigned char e = 0b10000110; // not standard C
```

a	0	0	1	1	1	1	0	1
b	1	0	0	1	0	1	1	0
&	0	0	0	1	0	1	0	0
|	1	0	1	1	1	1	1	1

yunglu@purdue.edu

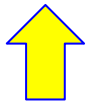
```
unsigned char a = 0x3D;  // 0011 1101
unsigned char b = 0x96;  // 1001 0110
unsigned char c = a & b; // 0001 0100 = 0X14
unsigned char d = a | b; // 1011 1111 = 0XBF
unsigned char e = 0b10000110; // not standard C
```

a	0	0	1	1	1	1	0	1
b	1	0	0	1	0	1	1	0
&	0	0	0	1	0	1	0	0
|	1	0	1	1	1	1	1	1

yunglu@purdue.edu

```
unsigned char a = 0x3D;  // 0011 1101
unsigned char b = 0x96;  // 1001 0110
unsigned char c = a & b; // 0001 0100 = 0X14
unsigned char d = a | b; // 1011 1111 = 0XBF
unsigned char e = 0b10000110; // not standard C
```

a	0	0	1	1	1	1	0	1
b	1	0	0	1	0	1	1	0
&	0	0	0	1	0	1	0	0
|	1	0	1	1	1	1	1	1



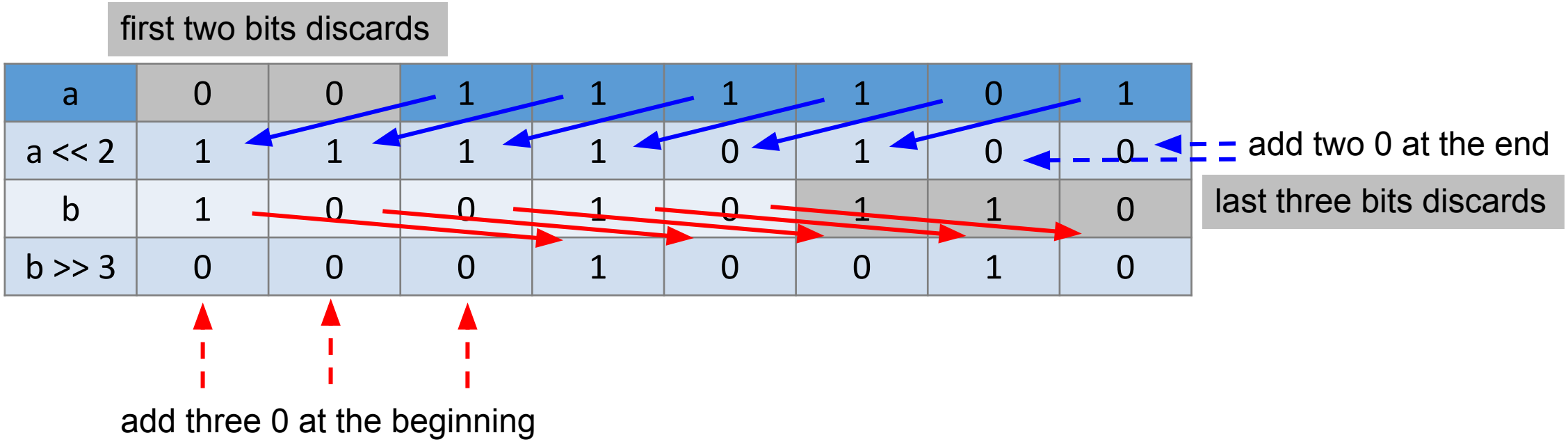
yunglu@purdue.edu

```
unsigned char a = 0x3D;  // 0011 1101
unsigned char b = 0x96;  // 1001 0110
unsigned char c = a & b; // 0001 0100 = 0X14
unsigned char d = a | b; // 1011 1111 = 0XBF
unsigned char e = 0b10000110; // not standard C
```

a	0	0	1	1	1	1	0	1
b	1	0	0	1	0	1	1	0
&	0	0	0	1	0	1	0	0
|	1	0	1	1	1	1	1	1

yunglu@purdue.edu

```
unsigned char a = 0x3D;    // 0011 1101
unsigned char b = 0x96;    // 1001 0110
unsigned char c = a << 2; // 1111 0100 = 0XF4
unsigned char d = b >> 3; // 0001 0010 = 0X12
```



yunglu@purdue.edu

Huffman Coding Increases File Sizes

- 1A 1m 1# 1 G 0 00 1c 1s 0 0 0
- + 4 bytes length
- +

Code Tree
Length of the original file
Code of the characters (data)

data	A	A	c	s	#	m	G	c	s	A	10 bytes	…
code	0 0	0 0	1 0	1 1	0 1 1 0	0 1 0	0 1 1 1	1 0	1 1	0 0	25 bytes	…

yunglu@purdue.edu

Use bits

	1	A	1	m	1	#	1	G	0	0	0	1	c	1	s	0	0	0
Hex		41		6D		23		47					63		73			

	1	A	1	m	1	#	1	G
Hex	1	0100 0001	1	0110 1101	1	0010 0011	1	0100 0111

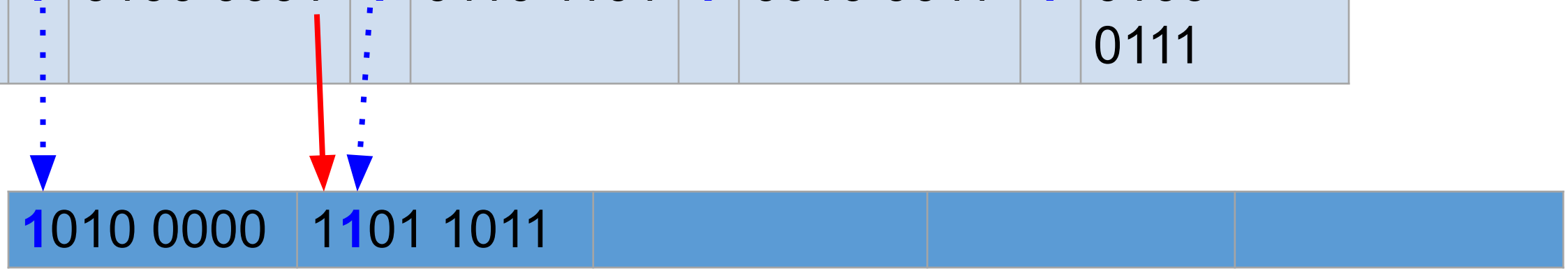
1010 0000				

yunglu@purdue.edu

Use bits

	1	A	1	m	1	#	1	G	0	0	0	1	c	1	s	0	0	0
Hex		41		6D		23		47					63		73			

	1	A	1	m	1	#	1	G
Hex	**1**	0100 0001	**1**	0110 1101	**1**	0010 0011	**1**	0100 0111

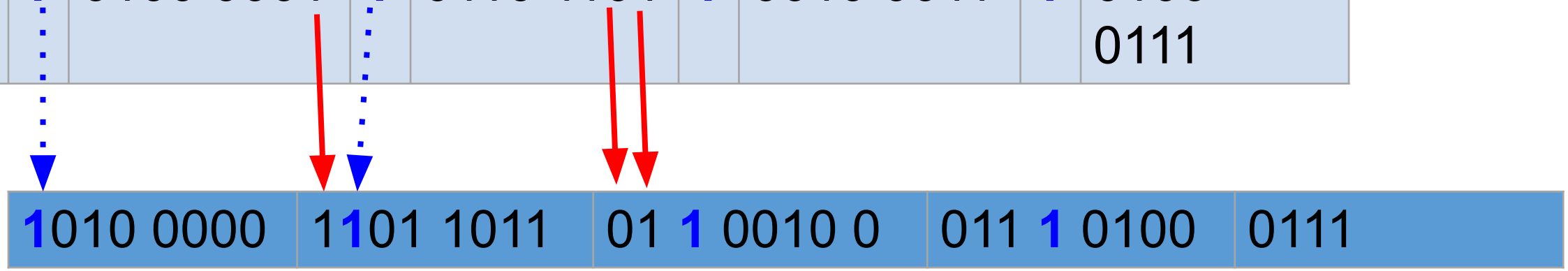


yunglu@purdue.edu

Use bits

	1	A	1	m	1	#	1	G	0	0	0	1	c	1	s	0	0	0
Hex		41		6D		23		47					63		73			

	1	A	1	m	1	#	1	G
Hex	1	0100 0001	1	0110 1101	1	0010 0011	1	0100 0111

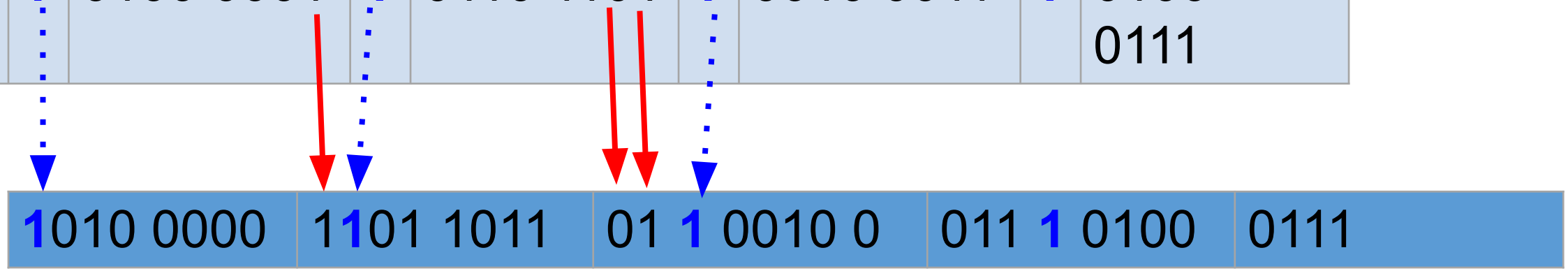


yunglu@purdue.edu

Use bits

	1	A	1	m	1	#	1	G	0	0	0	1	c	1	s	0	0	0
Hex		41		6D		23		47					63		73			

	1	A	1	m	1	#	1	G
Hex	1	0100 0001	1	0110 1101	1	0010 0011	1	0100 0111



yunglu@purdue.edu

Use bits

	1	A	1	m	1	#	1	G	0	0	0	1	c	1	s	0	0	0
Hex		41		6D		23		47					63		73			

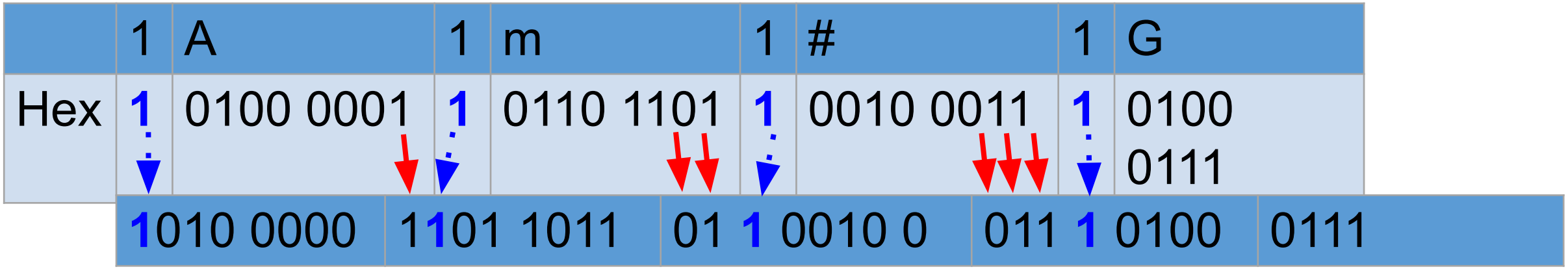
	1	A	1	m	1	#	1	G
Hex	1	0100 0001	1	0110 1101	1	0010 0011	1	0100 0111

yunglu@purdue.edu

	1	A	1	m	1	#	1	G
Hex	1	0100 0001	1	0110 1101	1	0010 0011	1	0100 0111
	1010 0000		1101 1011		01 1 0010 0		011 1 0100	0111

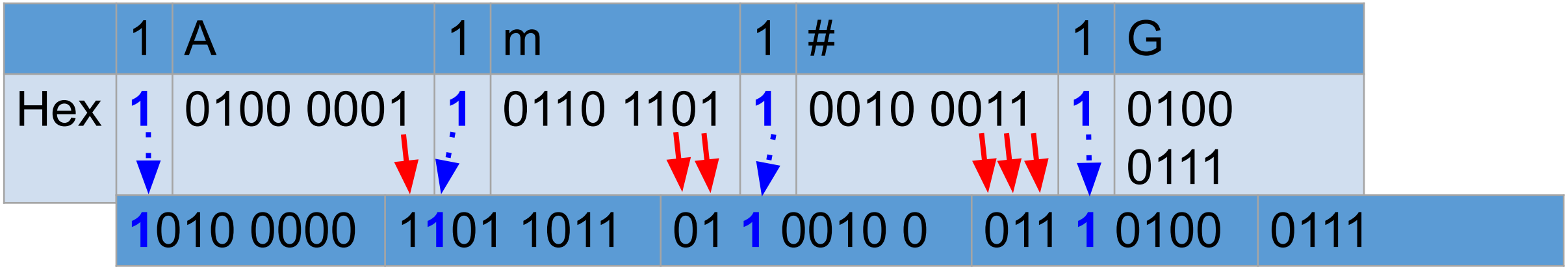
```
unsigned char a = 0x80;          // 1000 0000 control bit
unsigned char b = 'A' >> 1;      // 0010 0000
unsigned char onebyte = a | b; // 1010 0000 first byte
```

yunglu@purdue.edu



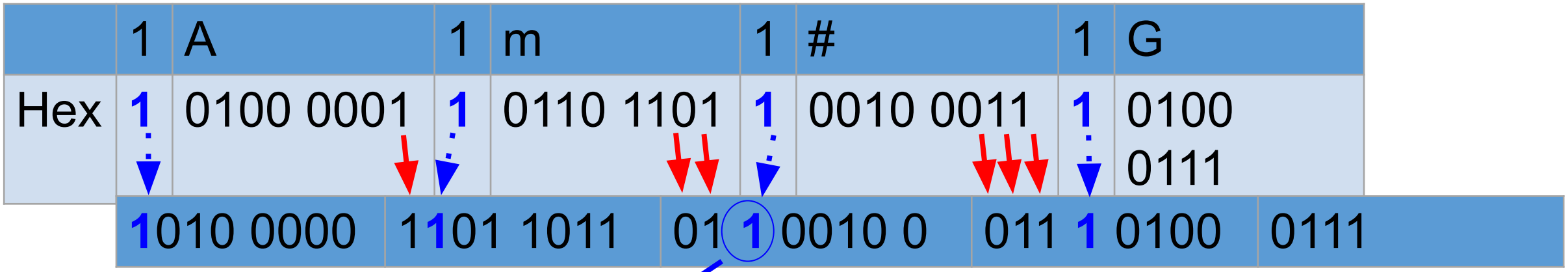

```
unsigned char a = 0x80;          // 1000 0000 control bit
unsigned char b = ‘A’ >> 1;      // 0010 0000
unsigned char onebyte = a | b; // 1010 0000 first byte
unsigned char c = ‘A’ & 0x01;  // 0000 0001 last bit of ‘A’
unsigned char d = c << 7;        // 1000 0000
// shift left 7 because ‘A’ 7 bits already in the first byte
unsigned char e = 0x80 >> 1; // 0100 0000, 1 + 7 = 8
onebyte = d | e | (‘m’ >> 2); // 1101 1011
```

yunglu@purdue.edu




```
unsigned char a = 0x80;          // 1000 0000 control bit
unsigned char b = ‘A’ >> 1;      // 0010 0000
unsigned char onebyte = a | b; // 1010 0000 first byte
unsigned char c = ‘A’ & 0x01;  // 0000 0001 last bit of ‘A’
unsigned char d = c << 7;        // 1000 0000
// shift left 7 because ‘A’ 7 bits already in the first byte
unsigned char e = 0x80 >> 1; // 0100 0000, 1 + 7 = 8
onebyte = d | e | (‘m’ >> 2); // 1101 1011
```

yunglu@purdue.edu




```
unsigned char g = 0x03 & 'm';   // 0000 0001, last 2 bits
unsigned char h = g << 6;       // 0100 0000
unsigned char i = 0x80 >> 2;    // 0010 0000
onebyte = h | i | ('#' >> 3);   // 0110 0100
```

yunglu@purdue.edu

	1	A	1	m	1	#	1	G
Hex	1	0100 0001	1	0110 1101	1	0010 0011	1	0100 0111
	1010 0000		1101 1011		01 1 0010 0		011 1 0100	0111

General Solution:

- 0x80 is the control of 1
- need to keep track how many bits to shift the control
- need to shift the characters right and left
- use **mask** to block unwanted bits

```
unsigned char c = 'A' & 0x01;  // last bit of 'A
```

- A mask blocks some information (such as cheek and forehead) and allows some information (such as eyes and mouth) to pass

yunglu@purdue.edu

	1	A	1	m	1	#	1	G	0	0	0	1	c	1	s	0	0	0
Hex		41		6D		23		47					63		73			

- Unused bits in the last byte (if any) are filled by zeros.

Code Tree
Length of the original file
Code of the characters (data)

data	A	A	c	s	#	m	G	c	s	A	10 bytes	…
code	0 0	0 0	1 0	1 1	0 1 1 0	0 1 0	0 1 1 1	1 0	1 1	0 0	25 bytes	…
	0 0 0 0 1 0 1 1				0 1 1 0 0 1 0 0		1 1 1 1 0 1 1 0					

yunglu@purdue.edu