

Computer Science 50

Introduction to Computer Science I

Harvard College

Week 7

David J. Malan

malan@post.harvard.edu

Valgrind

<http://valgrind.org/docs/manual/quick-start.html>

```
% valgrind -v --leak-check=full a.out
```

```
...
```

```
==23596== Invalid write of size 4
```

```
==23596==      at 0x80486DF: f (memory.c:22)
```

```
==23596==      by 0x80486FC: main (memory.c:29)
```

```
...
```

```
==23596== 40 bytes in 1 blocks are definitely lost in loss record 1 of 1
```

```
==23596==      at 0x4023595: malloc (vg_replace_malloc.c:149)
```

```
==23596==      by 0x80486D5: f (memory.c:21)
```

```
==23596==      by 0x80486FC: main (memory.c:29)
```

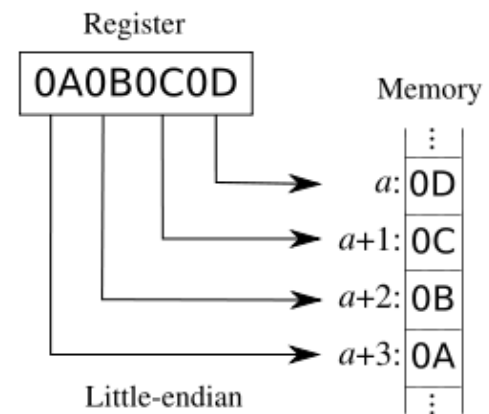
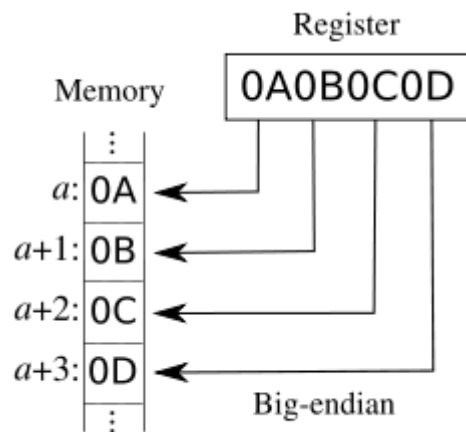
see
memory.c

Hexadecimal

0x01, ah ah ah....
0x02, ah ah ah...
0x03, ah ah ah...



Endianness



see
endian.c

Image from <http://en.wikipedia.org/wiki/Endianness>.

Bitwise Operators

- :: & bitwise AND
- :: | bitwise OR
- :: ^ bitwise XOR
- :: ~ ones complement
- :: << left shift
- :: >> right shift

Bitwise Operators

AND (&)

		B	
		0	1
A	0		
	1		

OR (|)

		B	
		0	1
A	0		
	1		

XOR (^)

		B	
		0	1
A	0		
	1		

ones complement (~)

A	0	
	1	

see
`binary.c`, `tolower.c`, `toupper.c`

Bitwise Operators

Swapping Values

```
int FOO = 1;  
int BAR = 4;
```

	// base-2 value in x	base-2 value in y
<code>int x = FOO;</code>	<code>// 001</code>	
<code>int y = BAR;</code>	<code>// 001</code>	100
<code>x = x ^ y;</code>	<code>// 001 ^ 100</code> <code>// 101</code>	100
<code>y = x ^ y;</code>	<code>// 101</code> <code>//</code>	101 ^ 100 001
<code>x = x ^ y;</code>	<code>// 101 ^ 001</code> <code>// 100</code>	001

see
`swap2.c`

Bitwise Operators

Swapping Values

```
int FOO = 1;  
int BAR = 4;
```

	// value in x	value in y
<code>int x = FOO;</code>	<code>// FOO</code>	
<code>int y = BAR;</code>	<code>// FOO</code>	BAR
<code>x = x ^ y;</code>	<code>// FOO ^ BAR</code>	BAR
<code>y = x ^ y;</code>	<code>// FOO ^ BAR</code>	$(\text{FOO} \wedge \text{BAR}) \wedge \text{BAR}$
	<code>//</code>	$\text{FOO} \wedge (\text{BAR} \wedge \text{BAR})$
	<code>//</code>	$\text{FOO} \wedge 0$
	<code>//</code>	FOO
<code>x = x ^ y;</code>	<code>// (FOO ^ BAR) ^ FOO</code>	FOO
	<code>// FOO ^ BAR ^ FOO</code>	
	<code>// FOO ^ FOO ^ BAR</code>	
	<code>// (FOO ^ FOO) ^ BAR</code>	
	<code>// 0 ^ BAR</code>	
	<code>// BAR</code>	

see
`swap2.c`

Hash Tables

Linear Probing

table[0]	
table[1]	
table[2]	
table[3]	
table[4]	
table[5]	
table[6]	
	⋮
table[24]	
table[25]	

Hash Tables

The Birthday Problem

In a room of n CS 50 students,
what's the probability that at least
two students share the same birthday?

Hash Tables

The Birthday Problem

$$\bar{p}(n) = 1 \cdot \left(1 - \frac{1}{365}\right) \cdot \left(1 - \frac{2}{365}\right) \cdots \left(1 - \frac{n-1}{365}\right) = \frac{365 \cdot 364 \cdots (365 - n + 1)}{365^n} = \frac{365!}{365^n (365 - n)!}$$

Hash Tables

The Birthday Problem

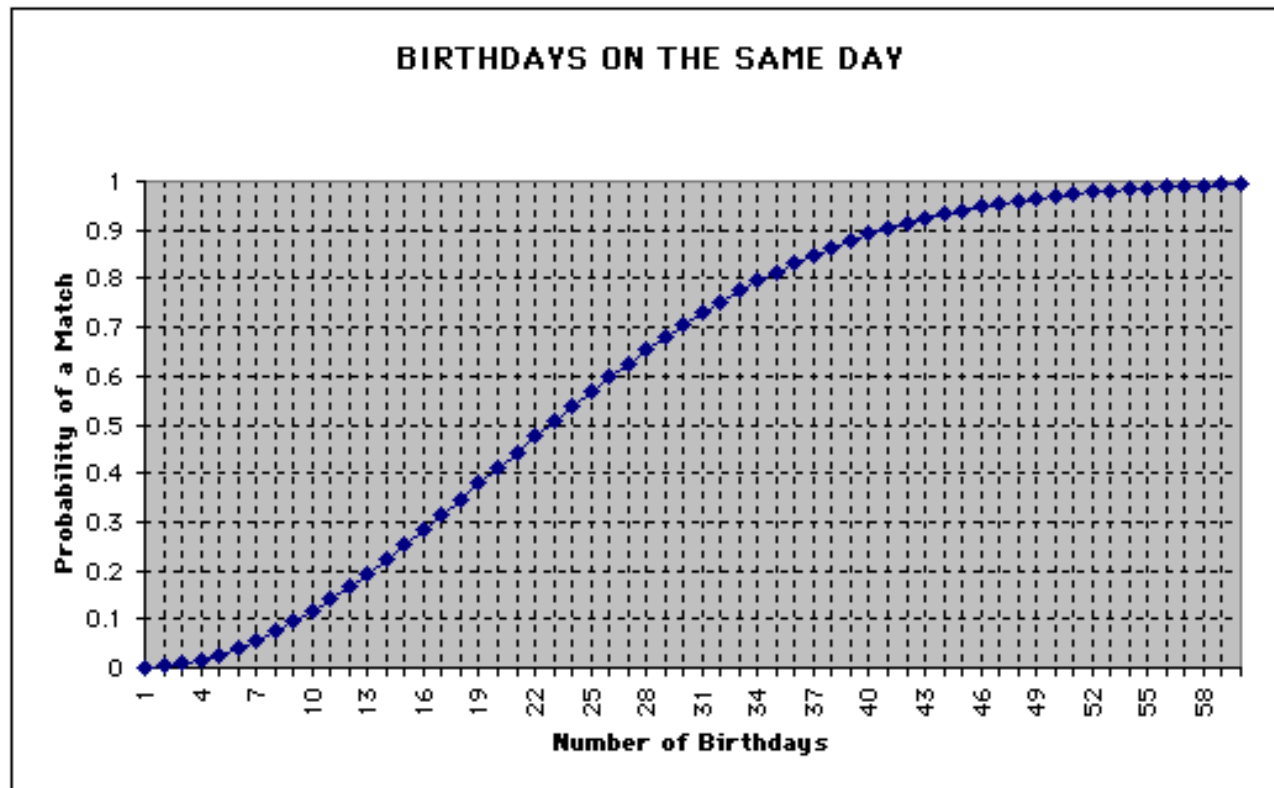


Image from <http://www.mste.uiuc.edu/reese/birthday/probchart.GIF>.

Hash Tables

Coalesced Chaining

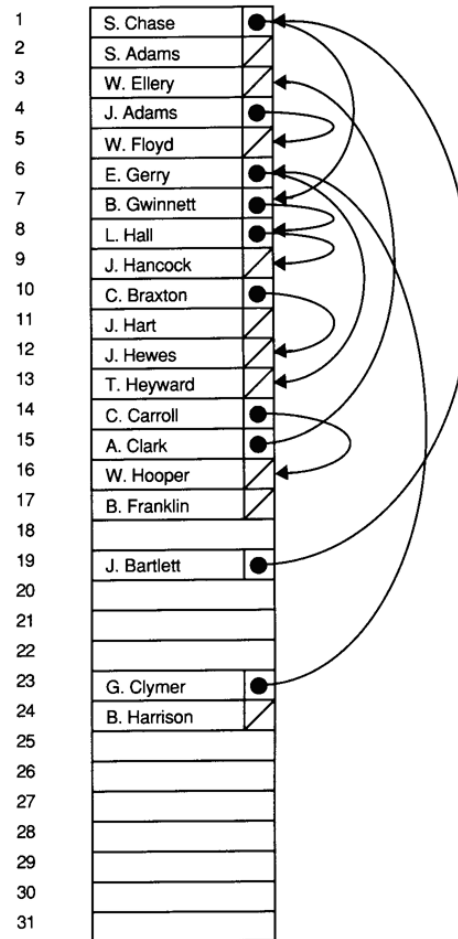


Figure from Lewis and Denenberg's *Data Structures & Their Algorithms*.

Hash Tables

Separate Chaining

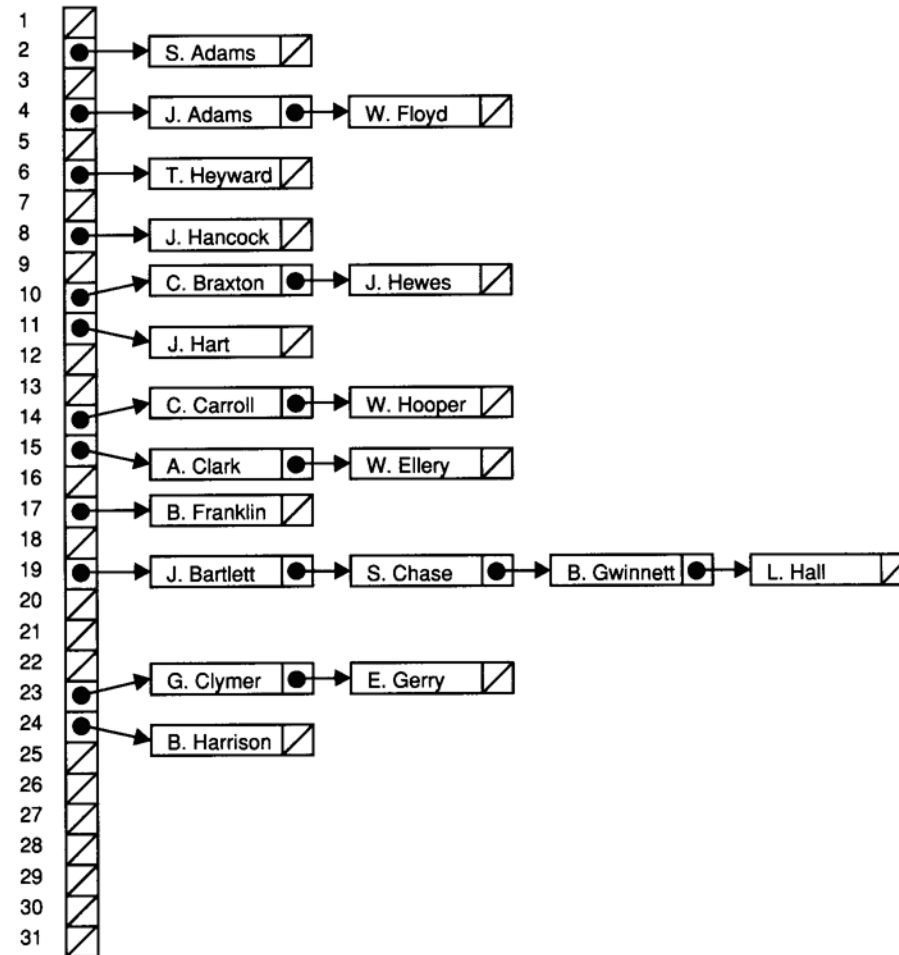


Figure from Lewis and Denenberg's *Data Structures & Their Algorithms*.

Trees

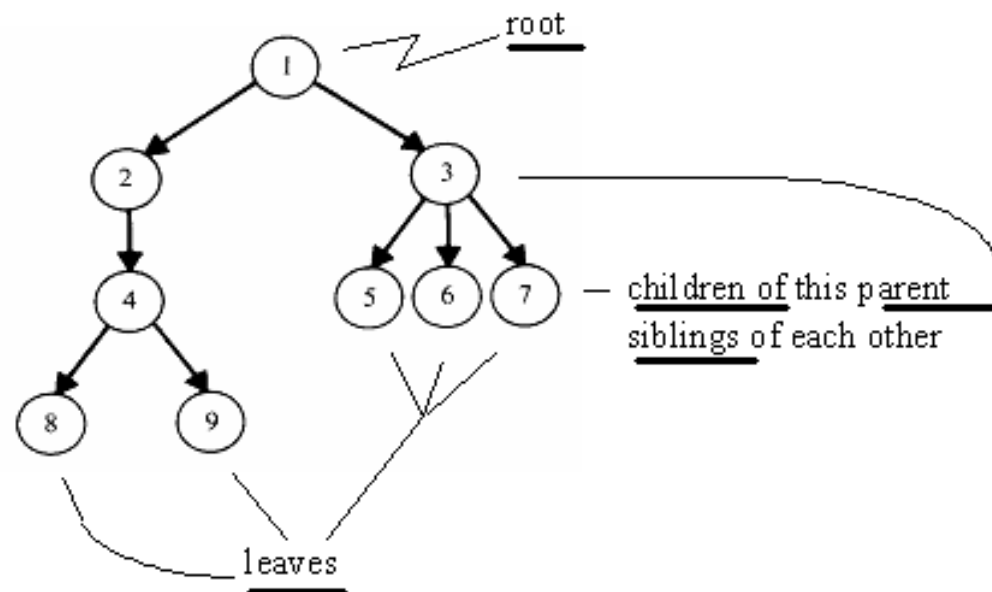
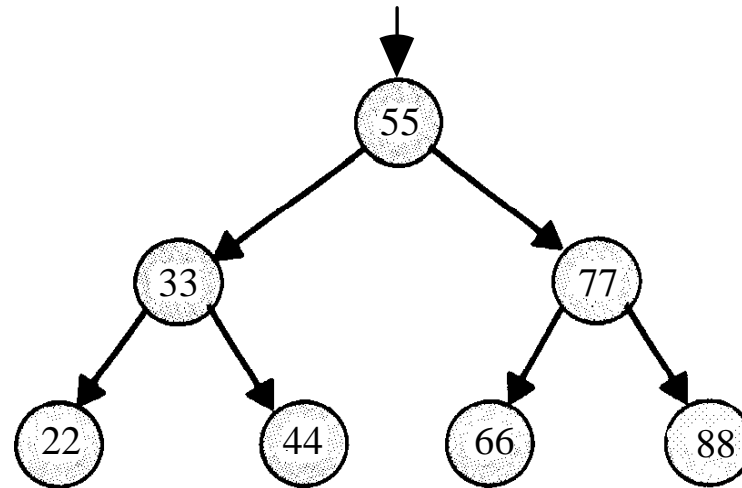
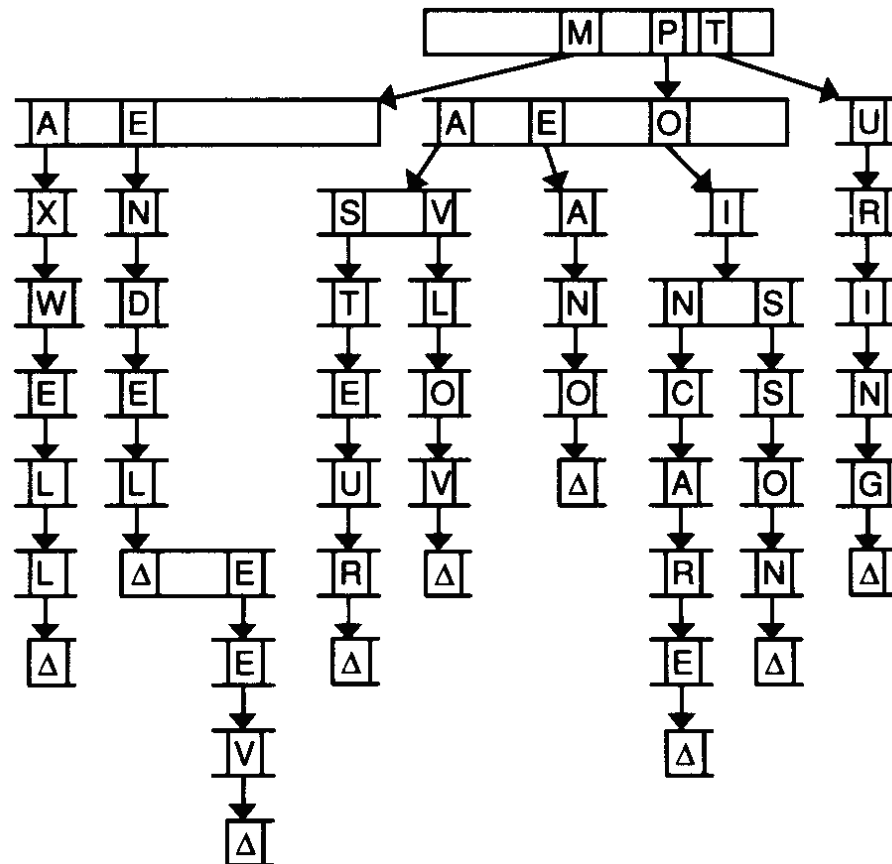


Figure by Larry Nyhoff.

Binary Search Trees

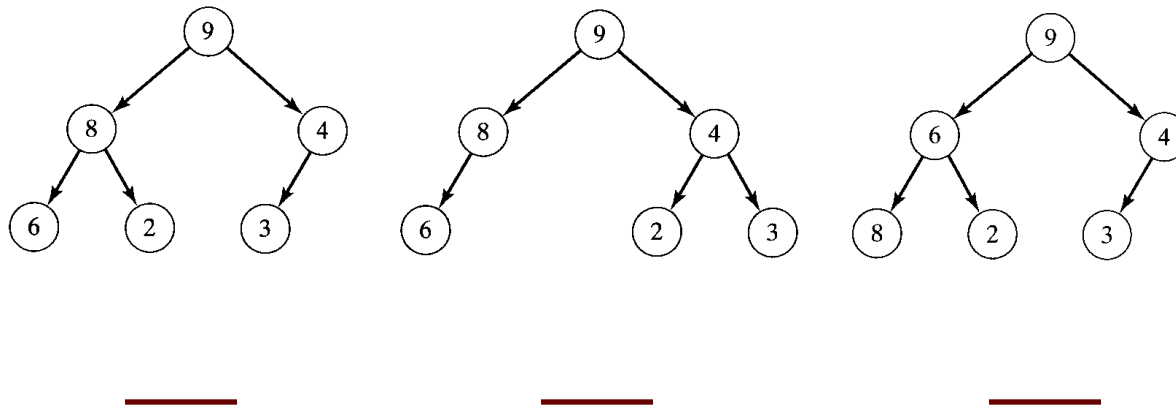


Tries

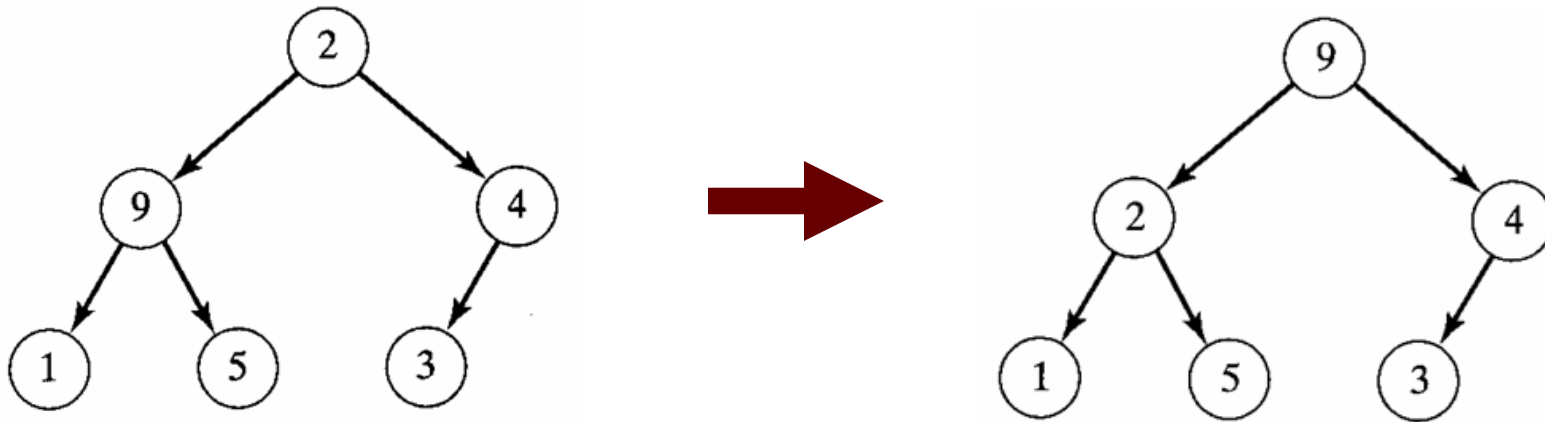


Heaps

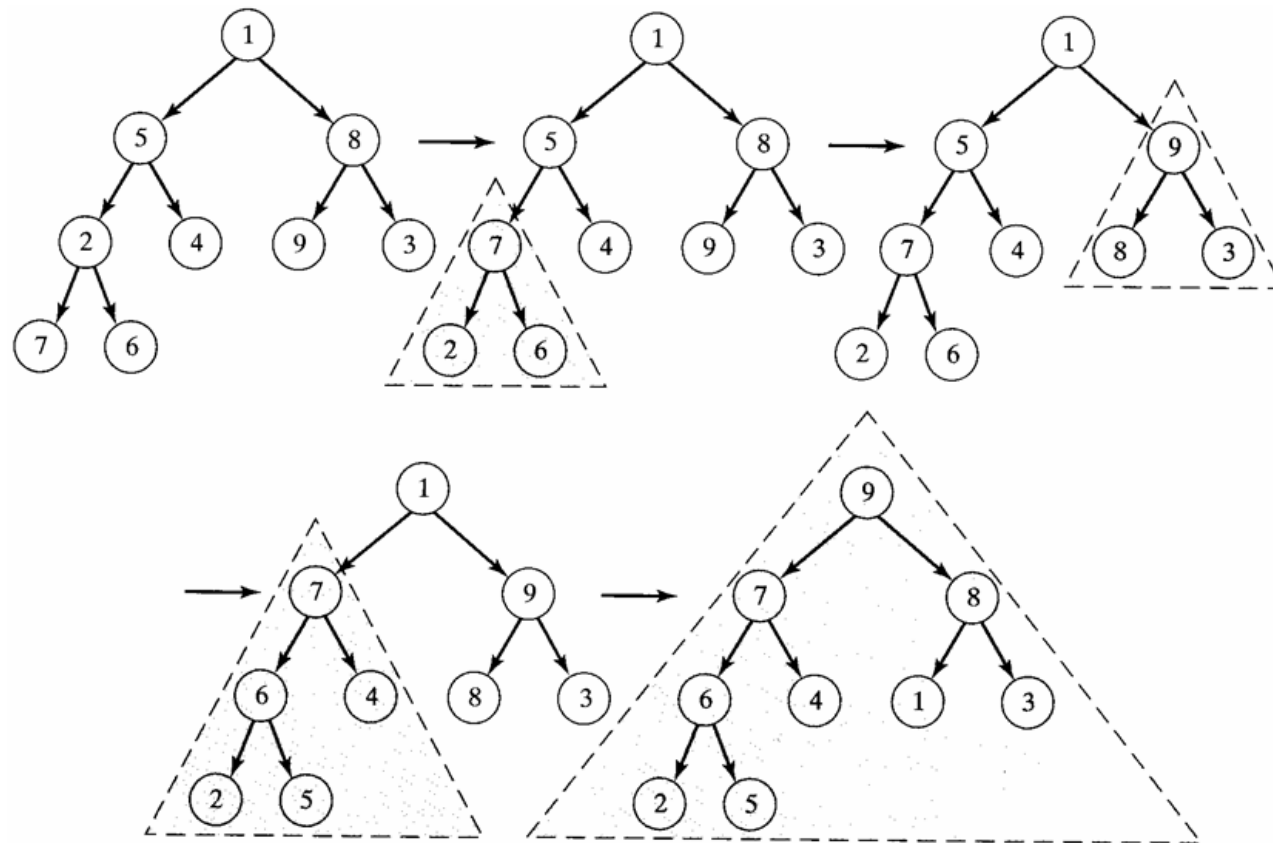
- :: A **heap** is a binary tree that
 - :: is **complete** (*i.e.*, every level of the tree is completely filled with nodes except for, perhaps, the bottommost level, whose nodes are in the leftmost locations)
 - :: satisfies the **heap-order property** (*i.e.*, each node's value is greater than or equal to that of each of its children, if any)



Heapifying an Almost Heap

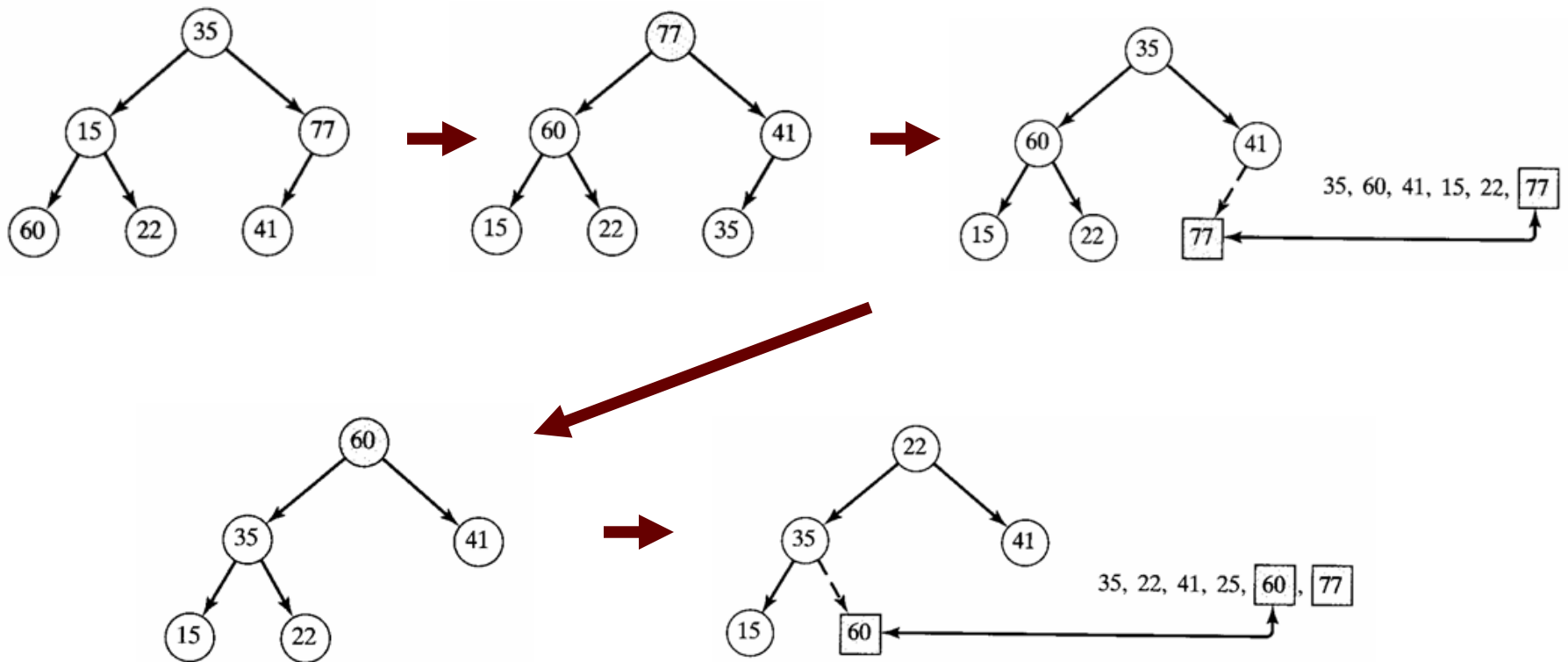


Heapifying a Binary Tree



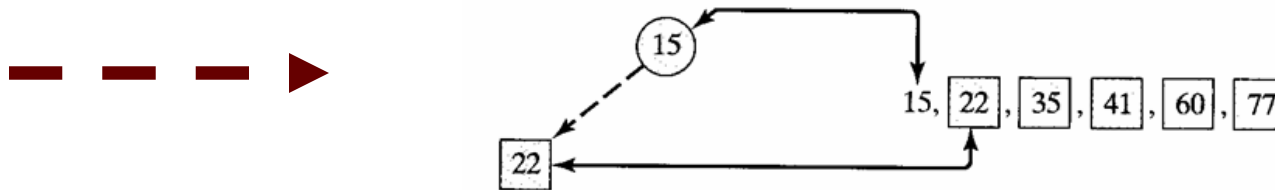
Heapsort

35 15 77 60 22 41



Heapsort

35 15 77 60 22 41



Morse Code

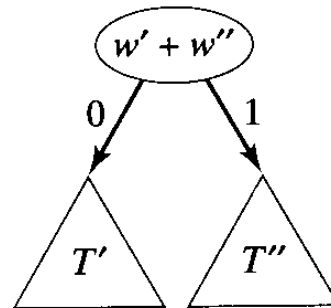
A	• —	U	• • —
B	— • • •	V	• • • —
C	— • — •	W	• — —
D	— • •	X	— • • —
E	•	Y	— • — —
F	• • — •	Z	— — • •
G	— — •		
H	• • • •		
I	• •		
J	• — — —		
K	— • —	1	• — — — —
L	• — • •	2	• • — — —
M	— —	3	• • • — —
N	— •	4	• • • • —
O	— — — —	5	• • • • •
P	• — — •	6	— • • • •
Q	— — • —	7	— — • • •
R	• — •	8	— — — • •
S	• • •	9	— — — — •
T	—	0	— — — — —

Image adapted from Wikipedia.

Huffman Coding

Immediate Decodability

- 1) Initialize a list of one-node binary trees containing weights w_1, w_2, \dots, w_n , one for each of the characters C_1, C_2, \dots, C_n .
- 2) Do the following $n - 1$ times:
 - a) Find two trees T' and T'' in this list with roots of minimal weight w' and w'' .
 - b) Replace these two trees with a binary tree whose root has weight $w' + w''$ and whose subtrees are T' and T'' ; label the pointers to these subtrees 0 and 1, respectively:



- 3) The code for character C_i is the bit string labeling the path from root to leaf C_i in the final binary tree.

Huffman Coding

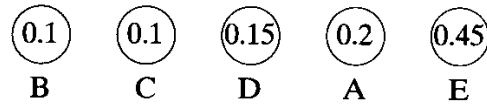
Example

“ECEABEADCAEDEEEEECEADEEEEEEDBAAEABDBBAAEAAAC
DDCCEABEEDCBEEDEAEFFFFFFAEEDBCEBEEADEAEEDAEB
CDEDEAEEDCEEAEFF”

character	A	B	C	D	E
frequency	0.2	0.1	0.1	0.15	0.45

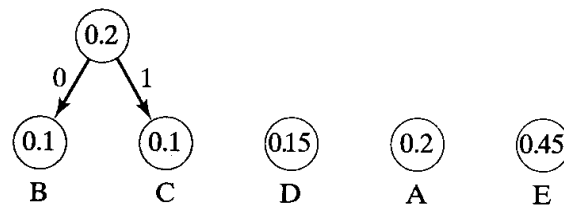
Huffman Coding

Example



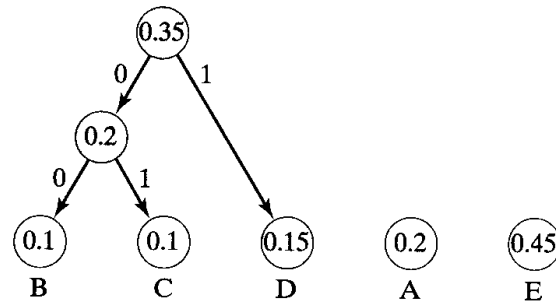
Huffman Coding

Example



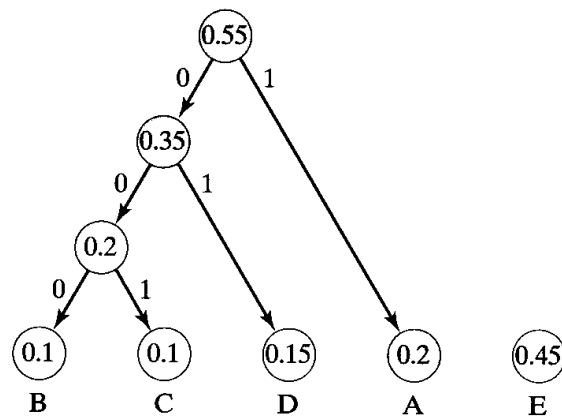
Huffman Coding

Example



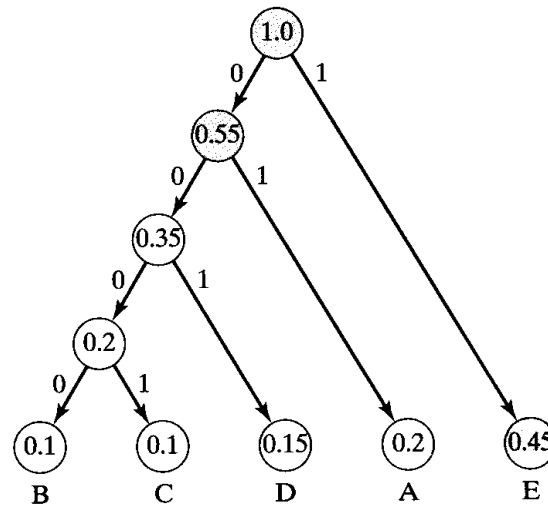
Huffman Coding

Example



Huffman Coding

Example



Huffman Coding

Example

Character	Huffman Code
A	
B	
C	
D	
E	

Huffman Coding

Problem?

0 1 0 1 0 1 1 0 1 0

Huffman Coding

In C

```
typedef struct node
{
    char symbol;
    int frequency;
    struct node *left;
    struct node *right;
}
node;
```

Computer Science 50

Introduction to Computer Science I

Harvard College

Week 7

David J. Malan

malan@post.harvard.edu