CS310: Advanced Data Structures and Algorithmns

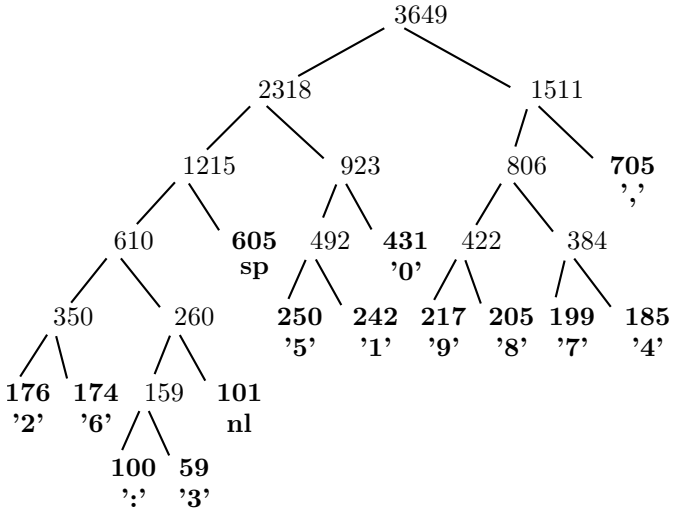
Spring 2014 Assignment 5 – Solution

Thanks to Prof. Betty O'Neil for parts of this solution.

1. a. Original order:

Char	:	' '	nl	,	0	1	2	3	4	5	6	7	8	9
Freq	100	605	101	705	431	242	176	59	185	250	174	199	205	217

The tree is as follows:



Resulting code table and analysis of it (not required):

character	code	frequency	total bits
:	000100	100	600
' '	001	605	1815
nl	00011	101	505
,	11	705	1410
0	011	431	1293
1	0101	242	968
2	00000	176	880
3	000101	59	354
4	1011	185	740
5	0100	250	1000
6	00001	174	870
7	1010	199	796
8	1001	205	820
9	1000	217	868

b. The most frequent is ' ', code = 11. The least frequent is '3', code = 000101.

c. With the coding scheme of 1, code "01, 08:"
0 → 011
04 → 011 1011

04: → 011 1011 000100
04: → 011 1011 000100 001
04: 1 → 011 1011 000100 001 0101
04: 19 → 011 1011 000100 001 0101 1000
04: 19, → 011 1011 000100 001 0101 1000 11 ← bit stream

0111 0110	0010 0001	0101 1000	1100 0000	← padded with 6 zero's
76	31	58	c0	

d. Decoding 11001110111100000:

11	001	11	011	11	00000
,	SP	,	0	,	2

e. Sorting by the character's code bitstring (like sorting char strings, first symbol, then second, ...) gives us the following table:

2	00000xxx
6	00001xxx
colon	000100xx
3	000101xx
newline	00011xxx
space	001xxxxx
5	0100xxxx
1	0101xxxx
0	011xxxxx
9	1000xxxx
8	1001xxxx
7	1010xxxx
4	1011xxxx
comma	11xxxxxx

So the first three codes are for '2', '6', and ':'. The '2' has three x's in its code pattern, for 3 don't-care bits, or $2^3 = 8$ bit patterns, followed by 8 for '6' and 4 for ':'

decode table:

index	value	
0	'2'	8 patterns for '2', from 00000 000 to 00000 111
1	'2'	
...		
7	'2'	
8	'6'	8 patterns for '6', from 00001 000 to 00001 111
9	'6'	
...		
15	'6'	
16	':'	4 patterns for ':', from 000100 00 to 000100 11
17	':'	
18	':'	
19	':'	
...		

2.

0041	0042	0031	0032	c2a9	0028	0029
A	B	1	2	©	(	)

3. a. 31 30 20 2d 33

b. 0031 0030 0020 002d 0033

c. 0031 0030 for "10" and 002d 0033 for "-3"

d. x: 0000000a in hex (32 bits for int, 0xa = 10, with 0 padding to left)
y: fffffffd in hex (32 bits for int, ffffffff = -1, fffffffe = -2, fffffffd = -3)

4. For 14.22:

a. A vertex is a 5 letter word, for example "bleed", "blood".

b. The edge shows that there is a transformation by a 1-letter change from the source word to the destination word, i.e. the two connected words differ by one character. Since it goes both ways, this can be considered an undirected graph.

c. Using the example, the graph looks like: $bleed \leftrightarrow blend \leftrightarrow blond \leftrightarrow blood$

d. As an adjaceny list:

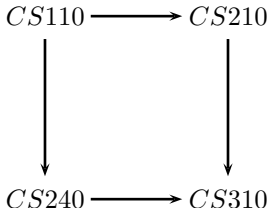
bleed: blend
blend: bleed, blond
blond: blend, blood
blood: blond

For 14.25:

a. A vertex is a course, for example CS110, or CS310, abbreviated 110, 310.

b. The edge represents a prerequisite relation between the source and destination node. This is a directed graph. The pre-requisite relation is one-way.

c. An example, made up for this class:



d. As an adjaceny list:

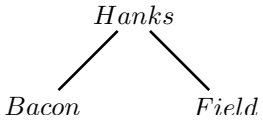
110: 210, 240
210: 310
240: 310

Repeat with 14.26:

a. A vertex is an actor, for example "Tom Hanks" or "Sally Field".

b. The edge relates one actor to another whose roles occurred in the same movie. This is an undirected graph.

c. The example:



d. As an adjaceny list:

Hanks: Bacon, Fields
Bacon: Hanks
Fields: Hanks

5. Here is the table:

A	B	C	D	E	Out
1	2	0	2	2	C
0	2	0	1	2	A
0	1	0	0	2	D
0	0	0	0	1	B
0	0	0	0	0	E

6. dfs from C:

visit node C befoe edges
visit edge CA

visit node A before edges
visit edge AB
visit node B before edges
visit edge BE
visit node E before edges
visit edge ED
visit node D before edges
visit node D after edges
visit node E after edges
visit edge BD
visit node B after edges
visit edge AD
visit node A after edges
visit edge CD
visit node C after edges

to get the topological sort order we look at the order in which the nodes were visited *after* edges, which is a topological sort order in reverse. In this example we get a topological sort order:

C A B E D

7. bfs from C:

Stage	Queue	Unseen
Begin	[C]	{A,B,D,E,F}
visit node C before edges	[]	{A,B,D,E,F}
visit edge CA	[A]	{B,D,E,F}
visit edge CB	[A,B]	{D,E,F}
visit node A before edges	[B]	{D,E,F}
visit edge AB	[B]	{D,E,F}
visit edge AD	[B,D]	{E,F}
visit node B before edges	[D]	{E,F}
visit edge BE	[D,E]	{F}
visit edge BF	[D,E,F]	{}
visit node D before edges	[E,F]	{}
visit edge DB	[E,F]	{}
visit edge DC	[E,F]	{}
visit node E before edges	[F]	{}
visit edge ED	[F]	{}
visit node F before edges	[]	{}