Mathematics for Computer Science final exam

소프트웨어학부 컴퓨터과학과
2016133 이유진

Software Design

Graph Algorithm

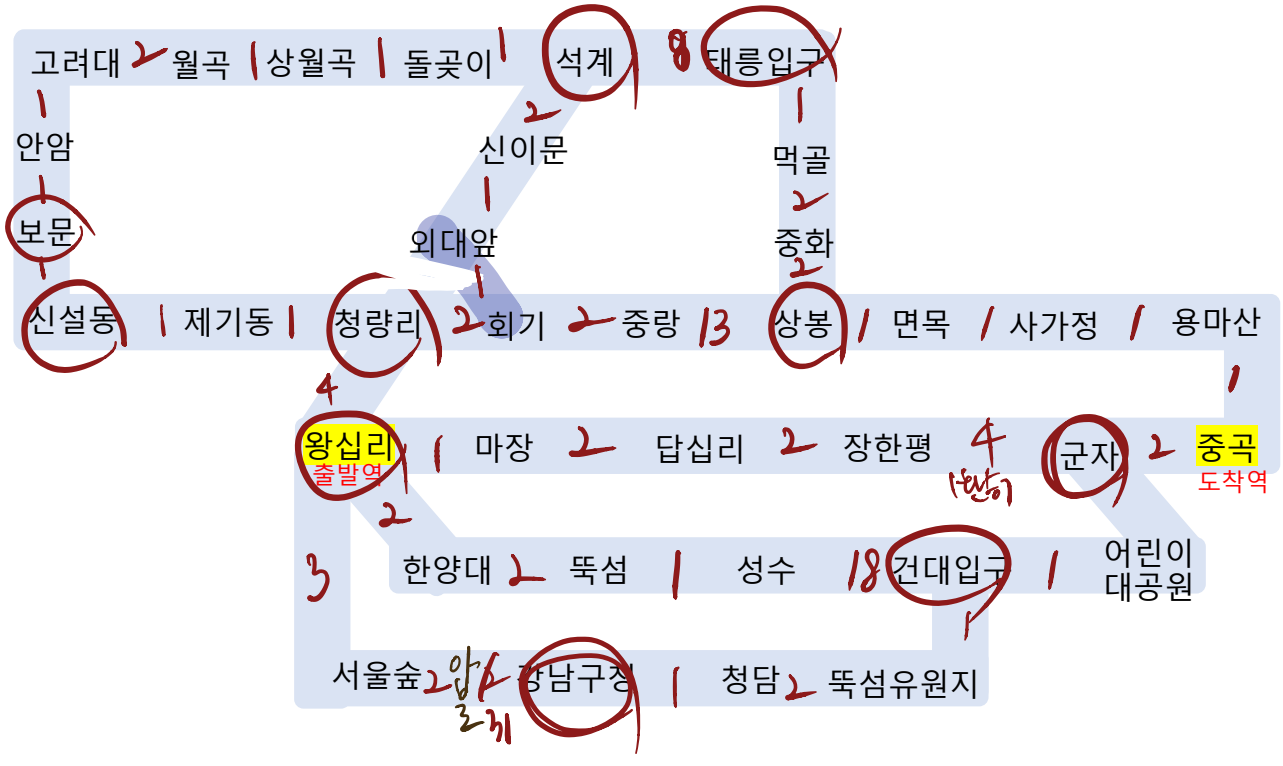
Step #1: Find two cases of shortest time path from a starting point to an arrival point

Comparison between two paths					
Case	Starting point	Arrival point	Item	Shortest time path found via naver-search	Shortest time path found via your method using Dijkstra's algorithm
Case #1	중곡	왕십리	Path	왕십리 – 마장 – 답십리 – 장한평 – 군자 - 중곡	
			Time		
Case #2	화곡	온수	Path	화곡 – 까치산 – 신정 – 목동 – 오목교 – 양평 – 영등포구청 – 영등포시장 – 신길 – 영등포 – 구로 – 구일 – 개봉 – 오류동 – 온수	
			Time		

Step #1-1: If you find any difference between two paths, explain why the difference arises.

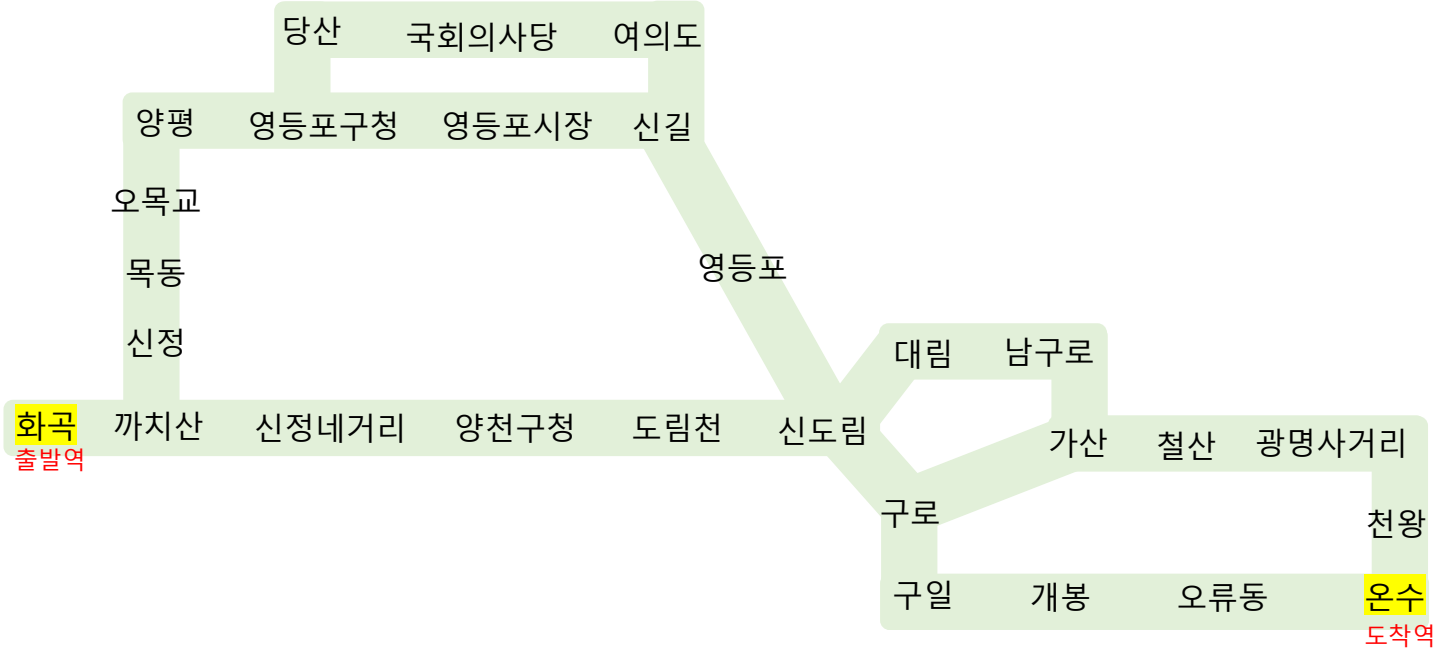
Step #2: Build two cases of simple & weighted graph model

Case #1



Step #2: Build two cases of simple & weighted graph model

Case #2



Step #3: Show the entire process of Dijkstra's algorithm for each case

Step #4: Let's consider subway accidents as below

Step #4-1: Let's assume subway accident stopping operation of a station

Step #4-2-1: Find 3 cases of shortest path in Case #1

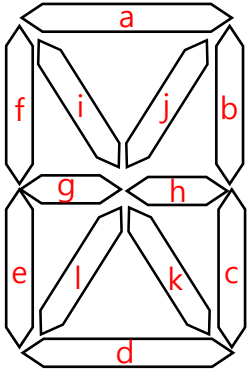
Step #4-2-2: Find 3 cases of shortest path in Case #2

Step #4-3: Tuning Dijkstra's algorithm

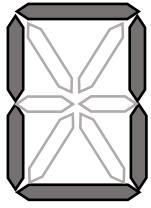
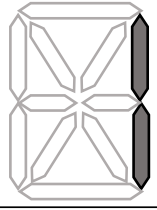
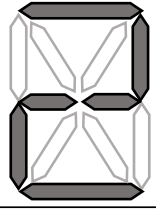
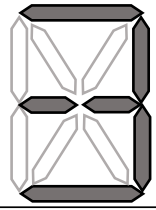
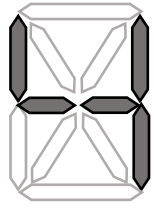
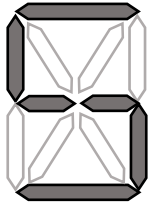
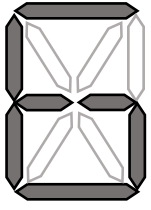
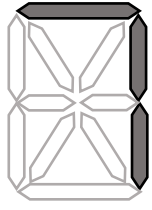
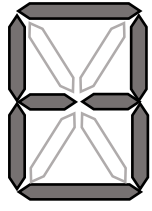
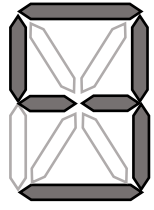
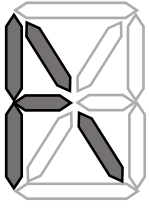
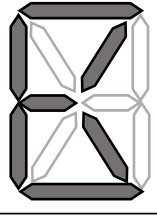
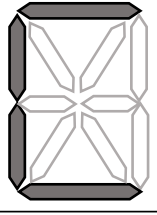
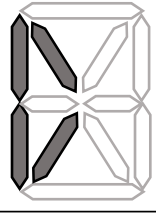
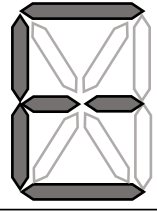
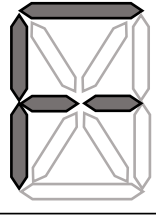
Hardware Design

Digital circuit based on boolean algebra

Step #1: Propose a way to represent hexadecimal number on 12-segment display



New display built on decimal number display

				
111111000000	011000000000	110110110000	111100110000	011001110000
				
101101110000	101111110000	111000000000	111111110000	111101110000
					
000011101010	100111100110	100111000000	000011001001	100111110000	100011110000

Step #2: Fill the below blank with your definition based on your proposal

Hexadecimal Number	Input (4-bit)				Output (Your definition)											
	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l
0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0
1	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1	1	0	0	0	0
3	0	0	1	1	1	1	1	1	0	0	1	1	0	0	0	0
4	0	1	0	0	0	1	1	0	0	1	1	1	0	0	0	0
5	0	1	0	1	1	0	1	1	0	1	1	1	0	0	0	0
6	0	1	1	0	1	0	1	1	1	1	1	1	0	0	0	0
7	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0
9	1	0	0	1	1	1	1	1	0	1	1	1	0	0	0	0
A	1	0	1	0	0	0	0	0	1	1	1	0	1	0	1	0
B	1	0	1	1	1	0	0	1	1	1	1	0	0	1	1	0
C	1	1	0	0	1	0	0	1	1	1	0	0	0	0	0	0
D	1	1	0	1	0	0	0	0	1	1	0	0	1	0	0	1
E	1	1	1	0	1	0	0	1	1	1	1	1	0	0	0	0
F	1	1	1	1	1	0	0	0	1	1	1	1	0	0	0	0

Step #3: Design BCHD(Binary Coded HexaDemical) to 12 Segment display decoder.

Output 'a'

Combinational logic design process

Step #1

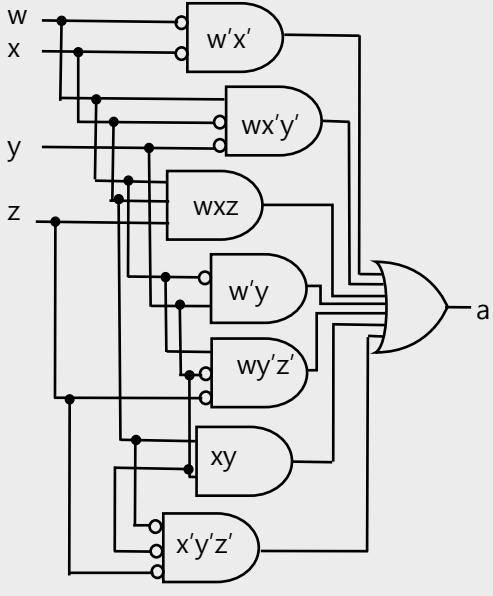
Hexadecimal Number	Input (4-bit)				Output
	w	x	y	z	a
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	1
A	1	0	1	0	0
B	1	0	1	1	1
C	1	1	0	0	1
D	1	1	0	1	0
E	1	1	1	0	1
F	1	1	1	1	1

Step #2

wx \ yz	00	01	11	10
00	1		1	1
01		1	1	1
11	1		1	1
10	1	1	1	

a = w'x'y'z'+w'x'yz+w'x'yz'
+w'xy'z+'wxyz+'wxyz'
+wxy'z' wxyz+wxyz'
+wx'y'z'+wx'y'z+wx'yz
=(wy'z'+x'y'z')+w'y+w'xz+xy
+(wx'y'+wxz)

Step #3



Step #3: Design BCHD(Binary Coded HexaDemical) to 12 Segment display decoder.

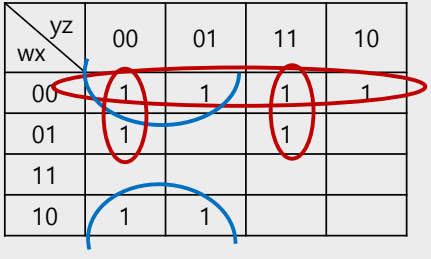
Output 'b'

Combinational logic design process

Step #1

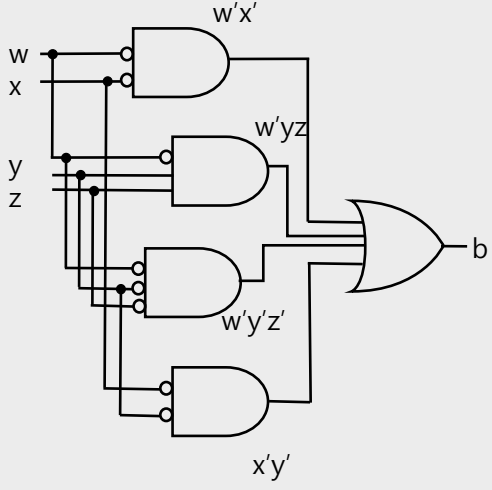
Hexadecimal Number	Input (4-bit)				Output
	w	x	y	z	b
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	1
A	1	0	1	0	0
B	1	0	1	1	0
C	1	1	0	0	0
D	1	1	0	1	0
E	1	1	1	0	0
F	1	1	1	1	0

Step #2



$b = w'x'y'z' + w'x'y'z + w'x'yz + w'x'yz'$
$+ w'xy'z' + w'xyz$
$+ wx'y'z' + wx'y'z$
$= w'x' + x'y' + w'y'z' + w'yz$

Step #3



Step #3: Design BCHD(Binary Coded HexaDemical) to 12 Segment display decoder.

Output 'c'

Combinational logic design process

Step #1

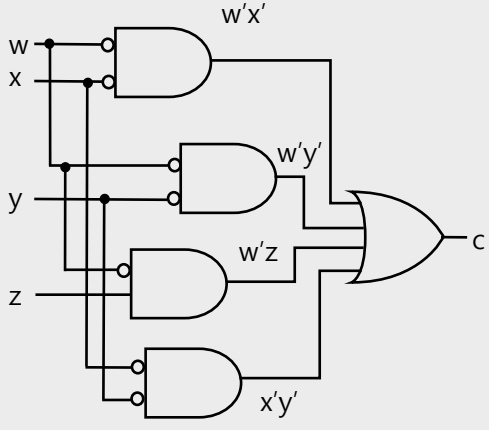
Hexadecimal Number	Input (4-bit)				Output
	w	x	y	z	c
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	1
A	1	0	1	0	0
B	1	0	1	1	0
C	1	1	0	0	0
D	1	1	0	1	0
E	1	1	1	0	0
F	1	1	1	1	0

Step #2

$c = w'x'y'z' + w'x'y'z + w'x'yz + w'xy'z' + w'xy'z + w'xyz + w'xyz' + wx'y'z' + wx'y'z$

$= x'y' + w'y' + w'z + w'x$

Step #3



Step #3: Design BCHD(Binary Coded HexaDemical) to 12 Segment display decoder.

Output 'd'

Combinational logic design process

Step #1

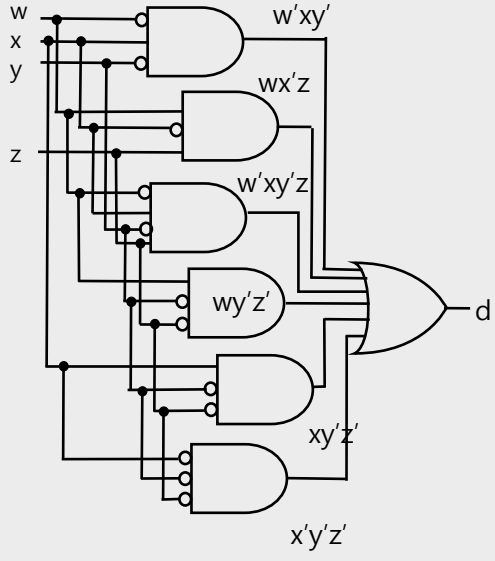
Hexadecimal Number	Input (4-bit)				Output
	w	x	y	z	d
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	0
8	1	0	0	0	1
9	1	0	0	1	1
A	1	0	1	0	0
B	1	0	1	1	1
C	1	1	0	0	1
D	1	1	0	1	0
E	1	1	1	0	1
F	1	1	1	1	0

Step #2

wx \ yz	00	01	11	10
00	1		1	1
01		1		1
11	1			1
10	1	1	1	

$$d = w'x'y'z' + w'x'yz + w'xy'z' + w'xy'z + w'xyz' + wxy'z' + wxyz + wx'y'z' + wx'y'z + wx'yz + wx'yz'$$
$$= w'x'y + (wy'z' + x'y'z') + w'xy'z + xyz' + wx'z$$

Step #3



Step #3: Design BCHD(Binary Coded HexaDemical) to 12 Segment display decoder.

Output 'e'

Combinational logic design process

Step #1

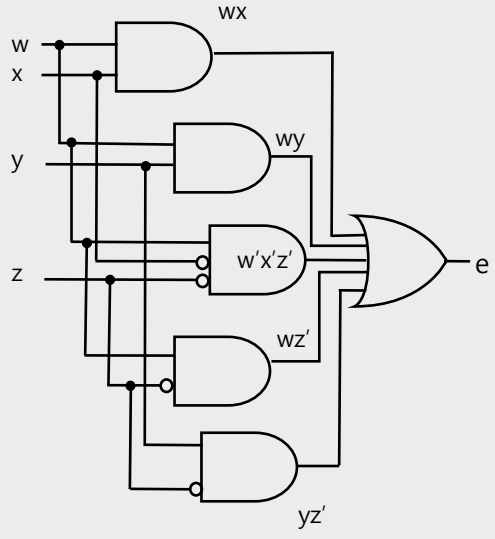
Hexadecimal Number	Input (4-bit)				Output
	w	x	y	z	e
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	1
7	0	1	1	1	0
8	1	0	0	0	1
9	1	0	0	1	0
A	1	0	1	0	1
B	1	0	1	1	1
C	1	1	0	0	1
D	1	1	0	1	1
E	1	1	1	0	1
F	1	1	1	1	1

Step #2

wx \ yz	00	01	11	10
00	1			1
01				1
11	1	1	1	1
10	1		1	1

$$e = w'x'y'z' + w'x'yz' + w'xyz' + wxy'z' + wxy'z + wxyz + wxyz' + wx'y'z' + wx'yz + wx'yz'$$
$$= w'x'z' + yz' + wz' + wx + wy$$

Step #3



Step #3: Design BCHD(Binary Coded HexaDemical) to 12 Segment display decoder.

Output 'f'

Combinational logic design process

Step #1

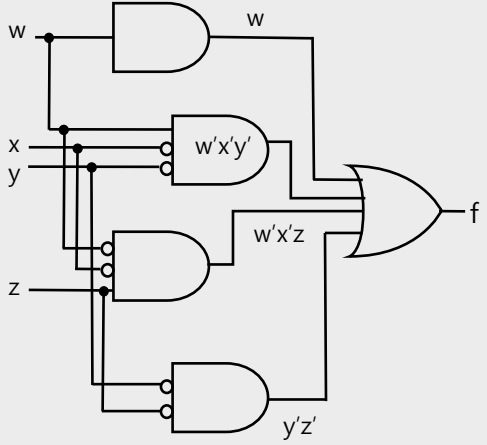
Hexadecimal Number	Input (4-bit)				Output
	w	x	y	z	f
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	0
8	1	0	0	0	1
9	1	0	0	1	1
A	1	0	1	0	1
B	1	0	1	1	1
C	1	1	0	0	1
D	1	1	0	1	1
E	1	1	1	0	1
F	1	1	1	1	1

Step #2

wx \ yz	00	01	11	10
00	1			
01	1	1		1
11	1	1	1	1
10	1	1	1	1

$$f = w'x'y'z' + w'xy'z' + w'xy'z + w'xyz' + wxy'z' + wxy'z + wxyz + wxyz' + wx'y'z' + wx'y'z + wx'yz + wx'yz'$$
$$= y'z' + (w'x'z' + w'x'y') + w$$

Step #3



Step #3: Design BCHD(Binary Coded HexaDemical) to 12 Segment display decoder.

Output 'g'

Combinational logic design process

Step #1

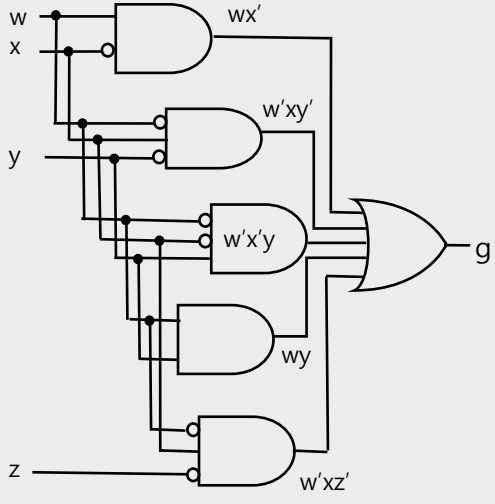
Hexadecimal Number	Input (4-bit)				Output
	w	x	y	z	g
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	0
8	1	0	0	0	1
9	1	0	0	1	1
A	1	0	1	0	1
B	1	0	1	1	1
C	1	1	0	0	0
D	1	1	0	1	0
E	1	1	1	0	1
F	1	1	1	1	1

Step #2

wx \ yz	00	01	11	10
00			1	1
01	1	1		1
11			1	1
10	1	1	1	1

$$g = w'x'yz + w'x'yz' + w'xy'z' + w'xy'z + w'xyz' + wxyz + wxyz' + wx'yz' + wx'y'z + wx'yz + wx'yz'$$
$$= w'x'y + (w'xy' + w'xz') + wx' + wy$$

Step #3



Step #3: Design BCHD(Binary Coded HexaDemical) to 12 Segment display decoder.

Output 'h'

Combinational logic design process

Step #1

Hexadecimal Number	Input (4-bit)				Output
	w	x	y	z	h
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	0
8	1	0	0	0	1
9	1	0	0	1	1
A	1	0	1	0	0
B	1	0	1	1	0
C	1	1	0	0	0
D	1	1	0	1	0
E	1	1	1	0	1
F	1	1	1	1	1

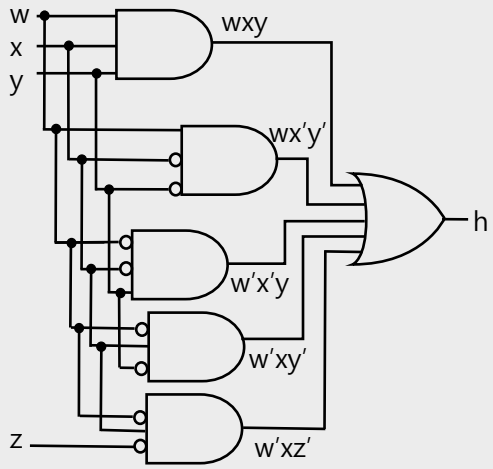
Step #2

wx \ yz	00	01	11	10
00			1	1
01	1	1		1
11			1	1
10	1	1		

$h = w'x'yz + w'x'yz' + w'xy'z' + w'xy'z + w'xyz' + wxyz + wxyz' + wx'y'z' + wx'y'z$

$= w'x'y + (w'xz' + w'xy') + wxy + wx'y'$

Step #3



Step #3: Design BCHD(Binary Coded HexaDemical) to 12 Segment display decoder.

Output 'i'

Combinational logic design process

Step #1

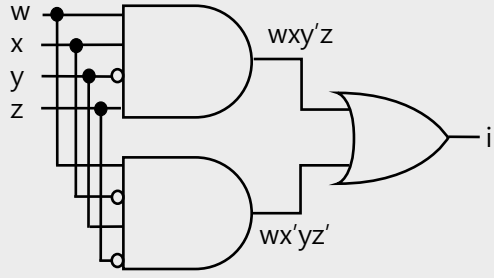
Hexadecimal Number	Input (4-bit)				Output
	w	x	y	z	i
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	0
A	1	0	1	0	1
B	1	0	1	1	0
C	1	1	0	0	0
D	1	1	0	1	1
E	1	1	1	0	0
F	1	1	1	1	0

Step #2

wx \ yz	00	01	11	10
00				
01				
11		1		
10				1

$i = wxy'z + wx'yz'$

Step #3



Step #3: Design BCHD(Binary Coded HexaDemical) to 12 Segment display decoder.

Output 'j'

Combinational logic design process

Step #1

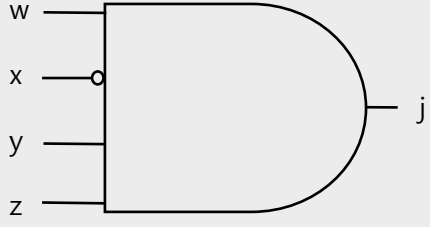
Hexadecimal Number	Input (4-bit)				Output
	w	x	y	z	j
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	0
A	1	0	1	0	0
B	1	0	1	1	1
C	1	1	0	0	0
D	1	1	0	1	0
E	1	1	1	0	0
F	1	1	1	1	0

Step #2

wx \ yz	00	01	11	10
00				
01				
11				
10			1	

$j = wx'yz$

Step #3



Step #3: Design BCHD(Binary Coded HexaDemical) to 12 Segment display decoder.

Output 'k'

Combinational logic design process

Step #1

Hexadecimal Number	Input (4-bit)				Output
	w	x	y	z	k
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	0
A	1	0	1	0	1
B	1	0	1	1	1
C	1	1	0	0	0
D	1	1	0	1	0
E	1	1	1	0	0
F	1	1	1	1	0

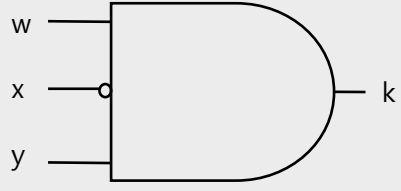
Step #2

wx \ yz	00	01	11	10
00				
01				
11				
10			1	1

$k = wx'yz + wx'yz'$

$= wx'y$

Step #3



Step #3: Design BCHD(Binary Coded HexaDemical) to 12 Segment display decoder.

Output 'l'

Combinational logic design process

Step #1

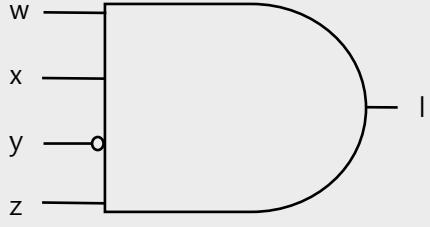
Hexadecimal Number	Input (4-bit)				Output
	w	x	y	z	l
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	0
A	1	0	1	0	0
B	1	0	1	1	0
C	1	1	0	0	0
D	1	1	0	1	1
E	1	1	1	0	0
F	1	1	1	1	0

Step #2

wx \ yz	00	01	11	10
00				
01				
11		1		
10				

l = wxy'z

Step #3



Step #4: Explain meaning of the designed BCHD to 12 segment display order