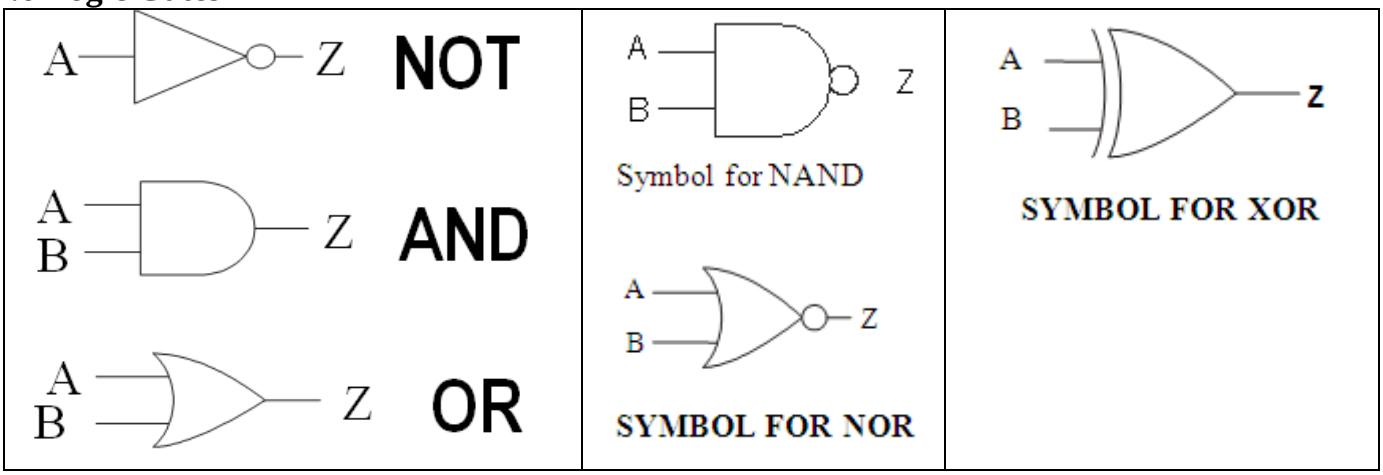
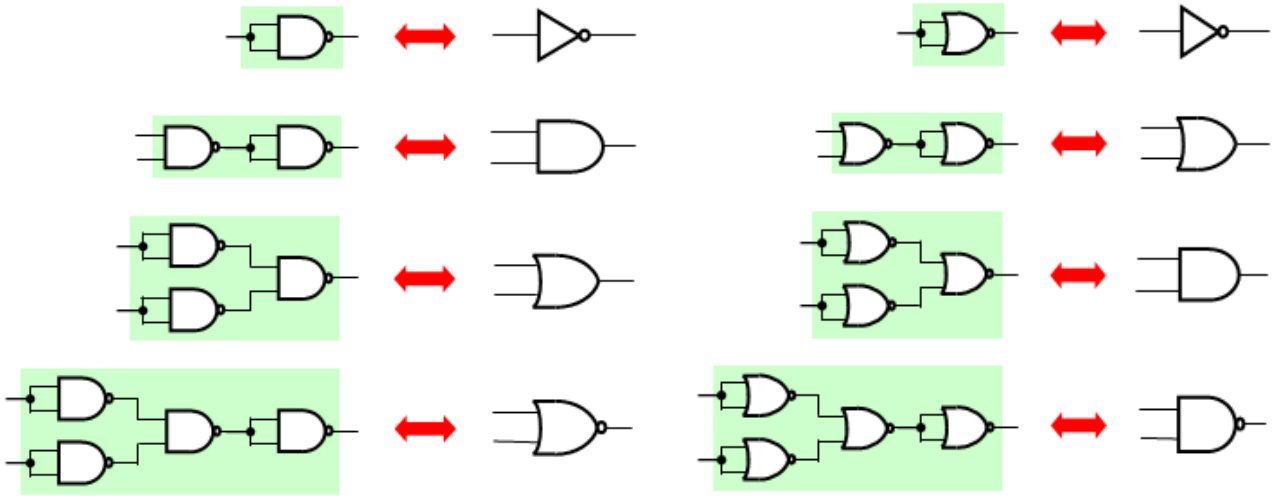
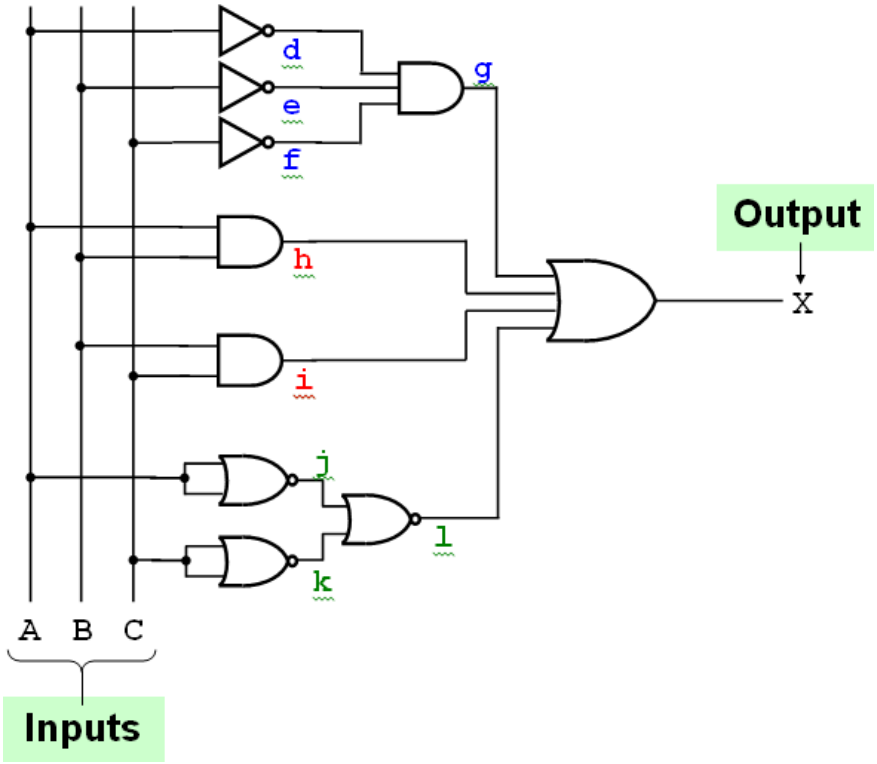
2.6 Logic Gates



Universal Property of NAND gates and NOR gates



Logic Circuits



Truth Table

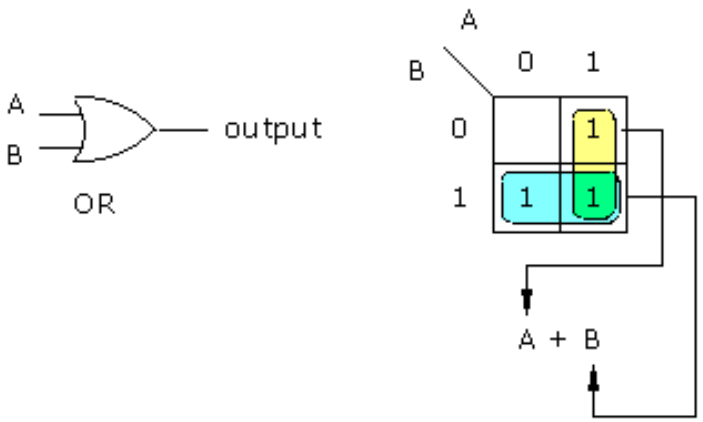
Input				Intermediate Nodes						Ouptut
A	B	C	d	e	f	g	h	i	j	k	l	X
0	0	0	1	1	1	1	0	0	1	1	0	1
0	0	1	1	1	0	0	0	0	1	0	0	0
0	1	0	1	0	1	0	0	0	1	1	0	0
0	1	1	1	0	0	0	0	1	1	0	0	1
1	0	0	0	1	1	0	0	0	0	1	0	0
1	0	1	0	1	0	0	0	0	0	0	1	1
1	1	0	0	0	1	0	1	0	0	1	0	1
1	1	1	0	0	0	0	1	1	0	0	1	1

K-maps

K-maps or Karnaugh maps provide an alternative way of simplifying logic circuits.
A **2-variable K-map** containing the minterm numbers is:

0	2
1	3

The map for a 2-input OR gate looks like this:



3-variable K-maps

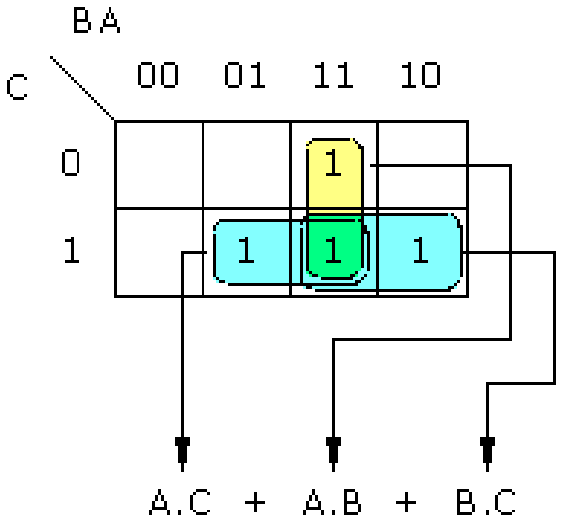
Here is the truth table for a 3-person majority voting system:

input C	*input B*	*input A*	*output*
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

A 3-variable K-map containing the minterm numbers is:

0	2	6	4
1	3	7	5

The vote is converted into a K-map, as follows:



With a little practice, this method is going to be quicker than the alternative, simplfiying the Boolean expression derived from the truth table:

$$A.B.\overline{C} + A.\overline{B}.C + \overline{A}.B.C + A.B.C$$

Note:

$$\overline{A}(\overline{B}.C + \overline{B}.\overline{C}) + \overline{A}.B.\overline{C}$$

This expression, you can't complete the truth table or Karnaugh map directly. First, you need to convert the statement into **sum of products**, or **SOP** form:

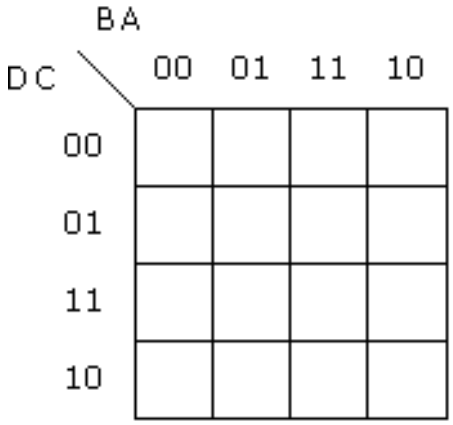
$$\overline{A}.\overline{B}.C + \overline{A}.\overline{B}.\overline{C} + \overline{A}.B.\overline{C}$$

4-variable maps

A 4-variable map will contain $2^4 = 16$ cells.

0	4	12	8
1	5	13	9
3	7	15	11
2	6	14	10

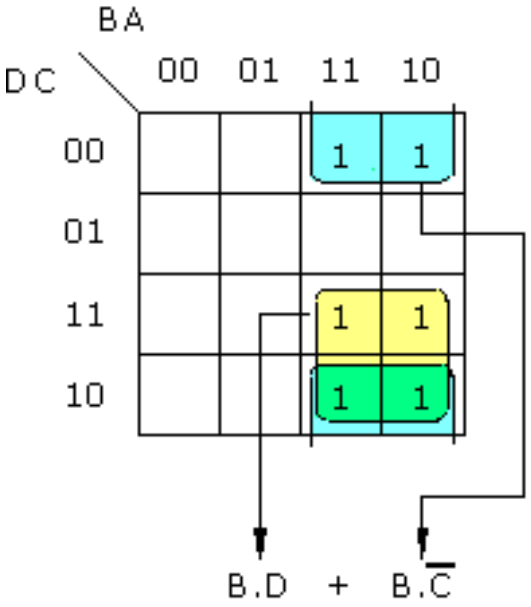
It is important to write the variable values along the columns and rows in Grey code:



To simplify the equation:

$$x = B.\overline{C}.\overline{D} + \overline{A}.B.\overline{C}.D + A.B.\overline{C}.D + \overline{A}.BCD + ABCD$$

The K-map becomes:



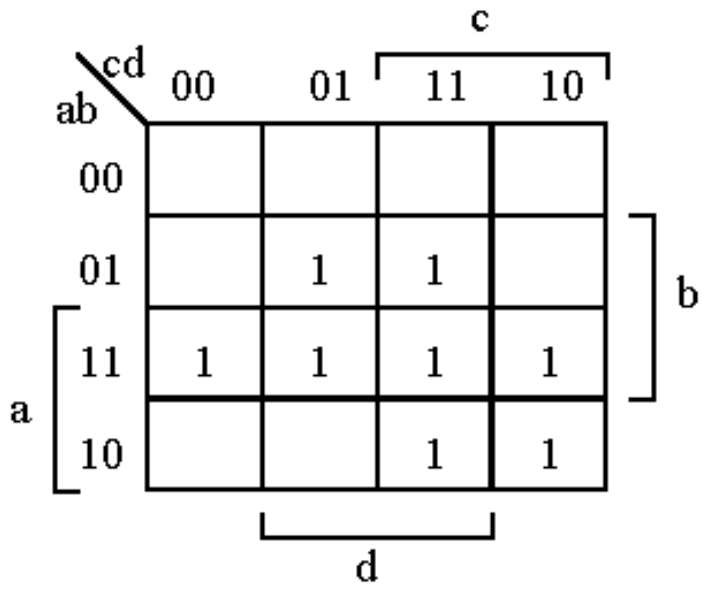
To give the simplest Boolean statement, you should put a circle round the maximum number of terms.

In this case, you can make two groups of four, one of which wraps around from top to bottom. You identify the two variables which remain constant in each group and eliminate the other two:

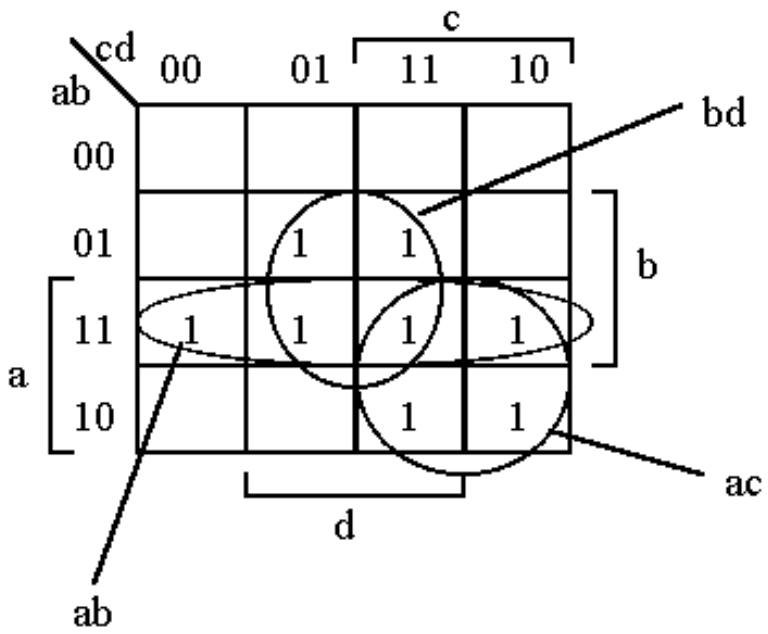
$$x = B.D + B.\overline{C}$$

RULE: Minimization is achieved by drawing the smallest possible number of circles, each containing the largest possible number of 1s.

Example: Consider
q = a'bc'd + a'bcd + abc'd' + abc'd + abcd + abcd' + ab'cd + ab'cd'
The following corresponds to the Boolean expression



Grouping the 1s together results in the following.

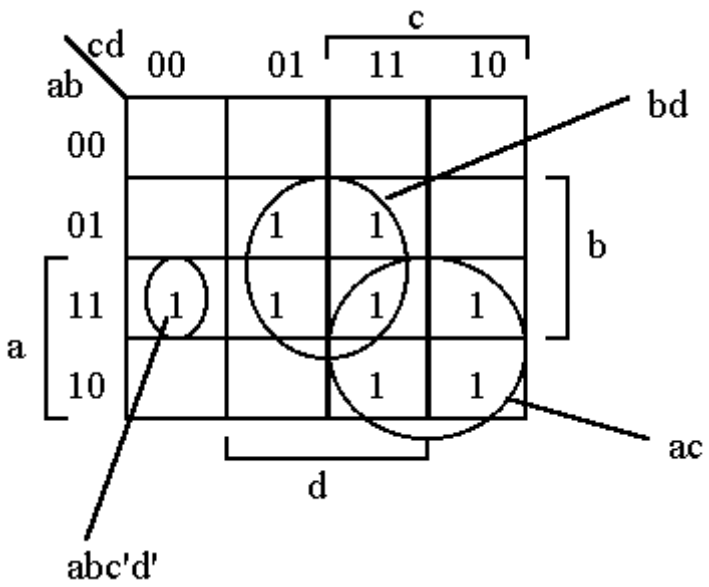


The expression for the groupings above is

q = bd + ac + ab

This expression requires 3 2-input **and** gates and 1 3-input **or** gate.

We could have accounted for all the 1s in the map as shown below, but that results in a more complex expression requiring a more complex gate.



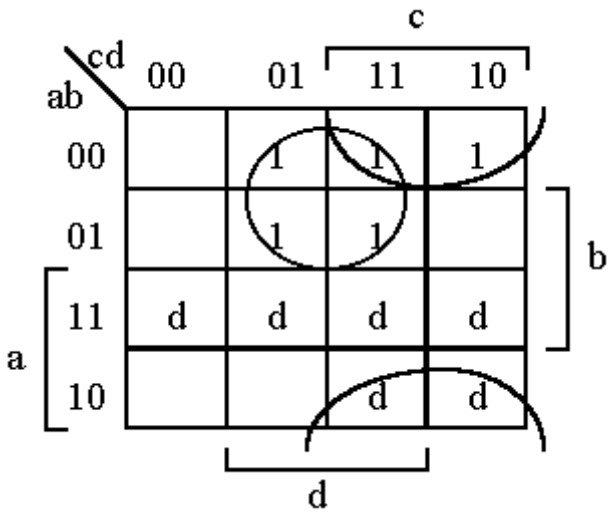
The expression for the above is **bd + ac + abc'd'**. This requires 2 2-input **and** gates, a 4-input **and** gate, and a 3 input **or** gate. Thus, one of the **and** gates is more complex (has two additional inputs) than required above. Two inverters are also needed.

Don't Cares

Sometimes we do not care whether a 1 or 0 occurs for a certain set of inputs. It may be that those inputs will never occur so it makes no difference what the output is. For example, we might have a bcd (binary coded decimal) code which consists of 4 bits to encode the digits 0 (0000) through 9 (1001). The remaining codes (1010 through 1111) are not used. If we had a truth table for the prime numbers 0 through 9, it would be

abcd	p
0000	0
0001	1
0010	1
0011	1
0100	0
0101	1
0110	0
0111	1
1000	0
1001	0
1010	d
1011	d
1100	d
1101	d
1110	d
1111	d

The ds in the above stand for "don't care", we don't care whether a 1 or 0 is the value for that combination of inputs because (in this case) the inputs will never occur.



The circle made entirely of 1s corresponds to the expression **a'd** and the combined 1 and d circle (actually a combination of arcs) is **b'c**. Thus, if the disallowed input 1011 did occur, the output would be 1 but if the disallowed input 1100 occurs, its output would be 0. The minimized expression is

p = a'd + b'c

Notice that if we had ignored the ds and only made a circle around the 2 1s, the resulting expression would have been more complex, **a'b'c** instead of **b'c**.