Discrete Mathematics Practical (Semester 3)

Reflexivity , Anti-Symmetry and Transitivity

In[110]:=

```
pairQ[{_, _}] := True;
pairQ[___] := False;
```

In[112]:=

```
relationQ[{__?pairQ}] := True;
relationQ[___] := False;
```

~Reflexivity

In[114]:=

```
reflexiveQ[R_?relationQ] := Module[{a, domain},
  domain = Union[Flatten[R, 1]];
  Catch[
   Do[If[! MemberQ[R, {a, a}], Throw[False]], {a, domain}];
   Throw[True]
  ]
 ]
```

In[115]:=

```
reflexiveQ[{{1, 1}, {2, 2}, {3, 3}, {4, 4}}]
```

Out[115]=

```
True
```

In[116]:=

```
reflexiveQ[{{1, 1}, {2, 2}, {3, 3}, {3, 4}}]
```

Out[116]=

```
False
```

Question - Create a relation R = {a, b : a∈A , b∈B and a divides b } (here input is set A)

In[117]:=

```
dividesRelation[A : {__Integer}] :=
 Select[Tuples[A, 2], Divisible[#[[2]], #[[1]]] &]
```

In[118]:=
```
B = {2, 3, 5, 8};
Tuples[B, 2]
dividesRelation[B]
```
Out[119]=
```
{{2, 2}, {2, 3}, {2, 5}, {2, 8}, {3, 2}, {3, 3}, {3, 5},
 {3, 8}, {5, 2}, {5, 3}, {5, 5}, {5, 8}, {8, 2}, {8, 3}, {8, 5}, {8, 8}}
```
Out[120]=
```
{{2, 2}, {2, 8}, {3, 3}, {5, 5}, {8, 8}}
```
In[121]:=
```
reflexiveQ[{{2, 2}, {2, 8}, {3, 3}, {5, 5}, {8, 8}}]
```
Out[121]=
```
True
```

~Anti-Symmetry

In[122]:=
```
AntiSymmetricQ[R__?relationQ] :=
 Module[{domain, a, b}, domain = Union[Flatten[R, 1]];
  Catch
   [Do[If[MemberQ[R, {a, b}] && MemberQ[R, {b, a}] && a ≠ b,
      Throw[False]], {a, domain}, {b, domain}];
   Throw[True]
  ]
 ]
```

In[123]:=
```
A = {{1, 1}, {2, 2}, {3, 3}};
B = {{1, 1}, {2, 2}, {3, 3}, {1, 4}, {4, 1}};
AntiSymmetricQ[A]
```
Out[125]=
```
True
```
In[126]:=
```
AntiSymmetricQ[B]
```
Out[126]=
```
False
```

~Transitivity

In[127]:=

```
TransitiveQ[R__?relationQ] :=
 Module[{domain , a , b , c}, domain = Union[Flatten[R, 1]];
  Catch
   [
   Do[If[MemberQ[R, {a, b}] && MemberQ[R, {b, c}] &&
      ! MemberQ[R, {a, c}],
     Throw[False]], {a, domain}, {b, domain} , {c, domain}];
   Throw[True]
  ]
 ]
```

In[128]:=

```
TransitiveQ[{{1, 2}, {2, 3}, {1, 3}}]
```

Out[128]=

```
True
```

In[129]:=

```
TransitiveQ[{{2, 2}, {2, 8}, {3, 3}, {5, 5}, {8, 8}}]
```

Out[129]=

```
True
```

PARTIAL ORDER

In[130]:=

```
partialOrderQ[R_?relationQ] :=
 reflexiveQ[R] && AntiSymmetricQ[R] && TransitiveQ[R]
```

In[131]:=

```
partialOrderQ[{{2, 2}, {2, 8}, {3, 3}, {5, 5}, {8, 8}}]
```

Out[131]=

```
True
```

In[132]:=

```
partialOrderQ[A]
```

Out[132]=

```
True
```

In[133]:=

```
partialOrderQ[B]
```

Out[133]=

```
False
```

In[134]:=
```
partialOrderQ[{{2, 2}, {2, 8}, {3, 3}, {5, 5}, {8, 8}}]
```
Out[134]=
```
True
```
In[135]:=
```
partialOrderQ[{{1, 1}, {2, 2}, {3, 3}, {4, 4}}]
```
Out[135]=
```
True
```

COVERING RELATION

In[136]:=
```
coversQ[R_?partialOrderQ, {x_, y_}] :=
 Module[{z, checkSet},
  Catch[
   If[x == y, Throw[False]];
   If[! MemberQ[R, {x, y}], Throw[False]];
   checkSet = Complement[Union[Flatten[R, 1]], {x, y}];
   Do[If[MemberQ[R, {x, z}] && MemberQ[R, {z, y}],
     Throw[False]],
    {z, checkSet}];
   Throw[True]
  ]
 ]
```
In[137]:=
```
r1 = {{1, 1}, {1, 2}, {1, 3}, {1, 4}, {2, 2}, {2, 3}, {2, 4}, {3, 3}, {3, 4}, {4, 4}};
coversQ[r1, {1, 2}]
```
Out[138]=
```
True
```
In[139]:=
```
coversQ[r1, {2, 2}]
```
Out[139]=
```
False
```
In[140]:=
```
coversQ[r1, {1, 4}]
```
Out[140]=
```
False
```
In[141]:=
```
coveringRelation[R_?partialOrderQ] :=
 Select[R, coversQ[R, #] &]
```
In[142]:=
```
coveringRelation[r1]
```
Out[142]=
```
{{1, 2}, {2, 3}, {3, 4}}
```

In[143]:=

```
P = {1, 2, 3, 4, 5};
Tuples[P, 2];
coveringRelation[dividesRelation[P]]
```

Out[145]=

```
{{1, 2}, {1, 3}, {1, 5}, {2, 4}}
```

◦Divisor Lattice

In[146]:=

```
divisorLattice[n_Integer] := dividesRelation[Divisors[n]]
divisorLattice[30]
```

Out[147]=

```
{{1, 1}, {1, 2}, {1, 3}, {1, 5}, {1, 6}, {1, 10}, {1, 15}, {1, 30}, {2, 2},
 {2, 6}, {2, 10}, {2, 30}, {3, 3}, {3, 6}, {3, 15}, {3, 30}, {5, 5}, {5, 10}, {5, 15},
 {5, 30}, {6, 6}, {6, 30}, {10, 10}, {10, 30}, {15, 15}, {15, 30}, {30, 30}}
```

In[148]:=

```
coveringRelation[divisorLattice[30]]
```

Out[148]=

```
{{1, 2}, {1, 3}, {1, 5}, {2, 6}, {2, 10}, {3, 6}, {3, 15}, {5, 10}, {5, 15}, {6, 30}, {10, 30}, {15, 30}}
```

◦Power Relation

In[149]:=

```
powerRelation[A : {__Integer}] :=
 Select[Tuples[Subsets[A], 2],
  Intersection[#[[1]], #[[2]]] == #[[1]] &]
```

In[150]:=

```
p1 = powerRelation[{1, 2}]
```

Out[150]=

```
{{{}, {}}, {{}, {1}}, {{}, {2}}, {{}, {1, 2}}, {{1}, {1}}, {{1}, {1, 2}}, {{2}, {2}}, {{2}, {1, 2}}, {{1, 2}, {1, 2}}}
```

In[151]:=

```
coversQ[p1, {{1}, {1, 2}}]
```

Out[151]=

```
True
```

In[152]:=

```
coveringRelation[p1]
```

Out[152]=

```
{{{}, {1}}, {{}, {2}}, {{1}, {1, 2}}, {{2}, {1, 2}}}
```

Minimal Elements

In[153]:=
```
minimalElements[R_?partialOrderQ, S_List] :=
 Module[{M, s, t},
  M = S;
  Do[
   Do[
    If[MemberQ[R, {t, s}], M = Complement[M, {s}]]
    , {t, Complement[S, {s}]}]
   , {s, S}];
  M
 ]
```

In[154]:=
```
minimalElements[divisorLattice[30], {2, 3, 6}]
```

Out[154]=
```
{2, 3}
```

In[155]:=
```
B = {1, 2, 3, 4, 5, 6};
div6 = dividesRelation[B]
```

Out[156]=
```
{{1, 1}, {1, 2}, {1, 3}, {1, 4}, {1, 5}, {1, 6}, {2, 2}, {2, 4}, {2, 6}, {3, 3}, {3, 6}, {4, 4}, {5, 5}, {6, 6}}
```

In[157]:=
```
minimalElements[div6, {2, 4, 6}]
```

Out[157]=
```
{2}
```

In[158]:=
```
minimalElements[div6, Range[6]]
```

Out[158]=
```
{1}
```

In[159]:=
```
minimalElements[divisorLattice[60], {10, 20, 15}]
```

Out[159]=
```
{10, 15}
```

In[160]:=
```
minimalElements[divisorLattice[60], {2, 20, 15}]
```

Out[160]=
```
{2, 15}
```

In[161]:=
```
minimalElements[divisorLattice[60], {2, 3, 5}]
```

Out[161]=
```
{2, 3, 5}
```

Maximal Elements

In[162]:=

```
maximalElements[R_?partialOrderQ, S_List] :=
 Module[{M, s, t},
  M = S;
  Do[
   Do[
    If[MemberQ[R, {s, t}], M = Complement[M, {s}]]
    , {t, Complement[S, {s}]}]
   , {s, S}];
  M
 ]
```

In[163]:=

```
maximalElements[divisorLattice[30], {2, 3, 6}]
```

Out[163]=

```
{6}
```

In[164]:=

```
maximalElements[div6, {2, 4, 6}]
```

Out[164]=

```
{4, 6}
```

In[165]:=

```
maximalElements[divisorLattice[60], {10, 20, 15}]
```

Out[165]=

```
{15, 20}
```

In[166]:=

```
maximalElements[div6, Range[6]]
```

Out[166]=

```
{4, 5, 6}
```

In[167]:=

```
maximalElements[divisorLattice[60], {2, 20, 15}]
```

Out[167]=

```
{15, 20}
```

In[168]:=

```
maximalElements[divisorLattice[60], {2, 3, 5}]
```

Out[168]=

```
{2, 3, 5}
```

UPPER BOUND

In[169]:=

```
upperBoundQ[R_?partialOrderQ, S_List, u_] := Module[{s},
  Catch[
   Do[If[! MemberQ[R, {s, u}], Throw[False]]
    , {s, S}];
   Throw[True]
  ]
 ]
```

In[170]:=

```
upperBoundQ[div6, {1, 2, 3}, 6]
```

Out[170]=

```
True
```

In[171]:=

```
upperBoundQ[div6, {1, 2, 3, 4}, 6]
```

Out[171]=

```
False
```

In[172]:=

In[173]:=

```
upperBounds[R_?partialOrderQ, S_List] :=
 Module[{domR, d , U = {}},
  domR = Union[Flatten[R, 1]];
  Do[If[upperBoundQ[R, S, d], AppendTo[U, d]]
   , {d, domR}];
  U
 ]
```

In[174]:=

```
upperBounds[div6, {1, 2, 3}]
```

Out[174]=

```
{6}
```

In[175]:=

```
upperBounds[divisorLattice[60], {1, 2, 5, 15}]
```

Out[175]=

```
{30, 60}
```

LEAST UPPER BOUND

In[176]:=

```
leastUpperBound[R_?partialOrderQ, S_List] :=
 Module[{U, M},
  U = upperBounds[R, S];
  M = minimalElements[R, U];
  If[Length[M] ≠ 1, Null , M[[1]]]
 ]
```

In[177]:=

```
leastUpperBound[div6, {1, 2}]
```

Out[177]=

```
2
```

LOWER BOUND

In[178]:=

```
lowerBoundQ[R_?partialOrderQ, S_List, l_] := Module[{s},
  Catch[
   Do[If[! MemberQ[R, {l, s}], Throw[False]],
    {s, S}];
   Throw[True]
  ]
 ]
```

In[179]:=

```
lowerBoundQ[div6, {1, 2, 3}, 1]
```

Out[179]=

```
True
```

In[180]:=

```
lowerBoundQ[div6, {1, 2, 3}, 4]
```

Out[180]=

```
False
```

In[181]:=

```
lowerBoundQ[div6, {2, 3, 4}, 1]
```

Out[181]=

```
True
```

In[182]:=

```
lowerBounds[R_?partialOrderQ, S_List] :=
 Module[{domR, d , L = {}},
  domR = Union[Flatten[R, 1]];
  Do[If[lowerBoundQ[R, S, d], AppendTo[L, d]]
   , {d, domR}];
  L
 ]
```

In[183]:=

```
lowerBounds[div6, {2, 6}]
```

Out[183]=

```
{1, 2}
```

GREATEST LOWER BOUND

In[184]:=

```
greatestLowerBound[R_?partialOrderQ, S_List] :=
 Module[{U, M},
  U = lowerBounds[R, S];
  M = maximalElements[R, U];
  If[Length[M] ≠ 1, "does not exist", M[[1]]]
 ]
```

In[185]:=

```
greatestLowerBound[div6, {3, 6}]
```

Out[185]=

```
3
```

In[186]:=

```
greatestLowerBound[div6, {3, 5, 6}]
```

Out[186]=

```
1
```

In[187]:=

```
greatestLowerBound[div6, {2, 4, 6}]
```

Out[187]=

```
2
```

In[188]:=

```
d66 = Complement[div6, {{1, 1}}]
```

Out[188]=

```
{{1, 2}, {1, 3}, {1, 4}, {1, 5}, {1, 6}, {2, 2}, {2, 4}, {2, 6}, {3, 3}, {3, 6}, {4, 4}, {5, 5}, {6, 6}}
```

Question 2.1 . Find join and meet :

In[189]:=

```
P = {1, 2, 3, 4, 5, 6, 7};
div7 = dividesRelation[P]
```

Out[190]=

```
{{1, 1}, {1, 2}, {1, 3}, {1, 4}, {1, 5}, {1, 6}, {1, 7},
 {2, 2}, {2, 4}, {2, 6}, {3, 3}, {3, 6}, {4, 4}, {5, 5}, {6, 6}, {7, 7}}
```

In[191]:=
```
leastUpperBound[div7, {3}]
```
Out[191]=
```
3
```
In[192]:=
```
greatestLowerBound[div7, {3}]
```
Out[192]=
```
3
```
In[193]:=
```
leastUpperBound[div7, {4, 6}]
```
In[194]:=
```
greatestLowerBound[div7, {4, 6}]
```
Out[194]=
```
2
```
In[195]:=
```
leastUpperBound[div7, {2, 3}]
```
Out[195]=
```
6
```
In[196]:=
```
greatestLowerBound[div7, {2, 3}]
```
Out[196]=
```
1
```
In[197]:=
```
leastUpperBound[div7, {2, 3, 6}]
```
Out[197]=
```
6
```
In[198]:=
```
greatestLowerBound[div7, {2, 3, 6}]
```
Out[198]=
```
1
```
In[199]:=
```
leastUpperBound[div7, {1, 5}]
```
Out[199]=
```
5
```
In[200]:=
```
greatestLowerBound[div7, {1, 5}]
```
Out[200]=
```
1
```

Question 2.2 .

In[201]:=

```
d60 = dividesRelation[{1, 2, 4, 5, 6, 12, 20, 30, 60}]
```

Out[201]=

```
{{1, 1}, {1, 2}, {1, 4}, {1, 5}, {1, 6}, {1, 12}, {1, 20}, {1, 30}, {1, 60}, {2, 2}, {2, 4}, {2, 6}, {2, 12},
 {2, 20}, {2, 30}, {2, 60}, {4, 4}, {4, 12}, {4, 20}, {4, 60}, {5, 5}, {5, 20}, {5, 30}, {5, 60}, {6, 6},
 {6, 12}, {6, 30}, {6, 60}, {12, 12}, {12, 60}, {20, 20}, {20, 60}, {30, 30}, {30, 60}, {60, 60}}
```

In[202]:=

```
greatestLowerBound[d60, {20, 30}]
```

Out[202]=

```
does not exist
```

In[203]:=

```
leastUpperBound[d60, {2, 5}]
```

HASSE DIAGRAMS

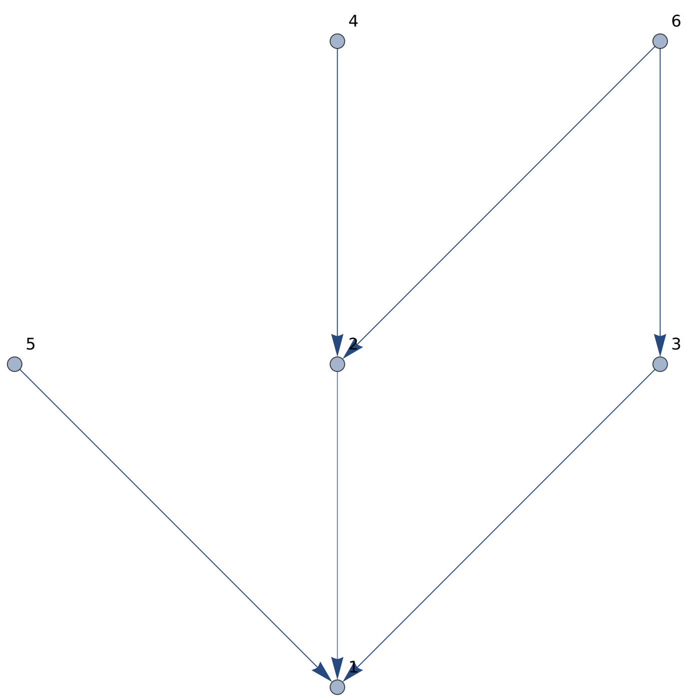
In[204]:=

```
hasseDiagram[R_?partialOrderQ] := Module[{edges},
   edges = coveringRelation[R] /. {a_, b_} → Rule[b, a];
   LayeredGraphPlot[edges, VertexLabeling → True]
  ]
```

In[205]:=

```
hasseDiagram[div6]
```

Out[205]=



In[206]:=

```
divisorLattice[20]
```

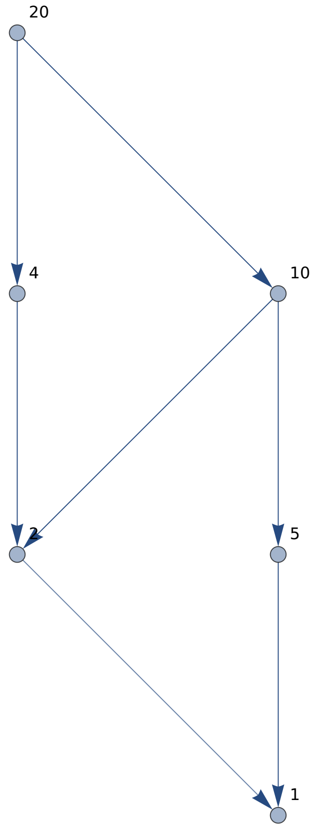
Out[206]=

{{1, 1}, {1, 2}, {1, 4}, {1, 5}, {1, 10}, {1, 20}, {2, 2}, {2, 4}, {2, 10}, {2, 20}, {4, 4}, {4, 20}, {5, 5}, {5, 10}, {5, 20}, {10, 10}, {10, 20}, {20, 20}}

In[207]:=

```
hasseDiagram[divisorLattice[20]]
```

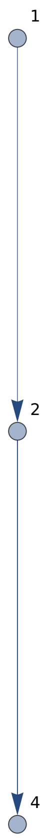
Out[207]=



In[208]:=

```
LayeredGraphPlot[{1 → 2, 2 → 4}, VertexLabeling → True]
```

Out[208]=



In[209]:=

In[210]:=

```
hasLUBs[R_?partialOrderQ] := Module[{domR, a, b},
  domR = Union[Flatten[R, 1]];
  Catch[
   Do[If[leastUpperBound[R, {a, b}] === Null , Throw[False]]
     , {a, domR}, {b, domR}];
   Throw[True]
  ]
 ]
```

In[211]:=

```
hasGLBs[R_?partialOrderQ] := Module[{domR, a, b},
  domR = Union[Flatten[R, 1]];
  Catch[
   Do[If[greatestLowerBound[R, {a, b}] === Null ,
      Throw[False]]
    , {a, domR}, {b, domR}];
   Throw[True]
  ]
 ]
```

In[212]:=

```
latticeQ[R_?partialOrderQ] := hasLUBs[R] && hasGLBs[R]
```

In[213]:=

```
div6
```

Out[213]=

```
{{1, 1}, {1, 2}, {1, 3}, {1, 4}, {1, 5}, {1, 6}, {2, 2}, {2, 4}, {2, 6}, {3, 3}, {3, 6}, {4, 4}, {5, 5}, {6, 6}}
```

In[214]:=

```
latticeQ[div6]
```

Out[214]=

```
False
```

In[215]:=

```
p1
```

Out[215]=

```
{{{}, {}}, {{}, {1}}, {{}, {2}}, {{}, {1, 2}}, {{1}, {1}}, {{1}, {1, 2}}, {{2}, {2}}, {{2}, {1, 2}}, {{1, 2}, {1, 2}}}
```

In[216]:=

```
latticeQ[p1]
```

Out[216]=

```
True
```

In[217]:=

```
latticeQ[divisorLattice[20]]
```

Out[217]=

```
True
```

Finding the following for a given boolean polynomial function.1. Representation of Boolean polynomial function and finding its value when the Boolean variable takes particular values over the Boolean Algebra{0,1}.2. Display in table form of all possible values of Boolean polynomial function over the Boolean {0,1}.

Logic Operators ((&& , ∧) AND ,(|| , ∨) OR and (¬ , ! ,)NOT)

In[218]:=
```
True || False
```
Out[218]=
```
True
```
In[219]:=
```
True || True
```
Out[219]=
```
True
```
In[220]:=
```
False || True
```
Out[220]=
```
True
```
In[221]:=
```
False || False
```
Out[221]=
```
False
```
In[222]:=
```
True ∨ False
```
Out[222]=
```
True
```
In[223]:=
```
True && True
```
Out[223]=
```
True
```
In[224]:=
```
True && False
```
Out[224]=
```
False
```
In[225]:=
```
False && True
```
Out[225]=
```
False
```

In[226]:=

```
False && False
```

Out[226]=

```
False
```

In[227]:=

```
False ∧ True
```

Out[227]=

```
False
```

In[228]:=

```
! True
```

Out[228]=

```
False
```

In[229]:=

```
¬ False
```

Out[229]=

```
True
```

In[230]:=

```
True && ¬ True
```

Out[230]=

```
False
```

In[231]:=

```
False ∨ ¬ False
```

Out[231]=

```
True
```

In[232]:=

```
¬ True ∧ ¬ False
```

Out[232]=

```
False
```

Representing Boolean Functions

1. $f(x, y, z) = xy + yz + zx$

In[233]:=

```
f[x_, y_, z_] := (x ∧ y) ∨ (y ∧ z) ∨ (z ∧ x);
f[p, q, r]
```

Out[234]=

```
(p && q) || (q && r) || (r && p)
```

In[235]:=

```
f[True, False , True]
```

Out[235]=

```
True
```

In[236]:=

```
f[True, False, True]
```

Out[236]=

```
True
```

In[237]:=

```
f[True, q, r]
```

Out[237]=

```
q || (q && r) || r
```

In[238]:=

```
f[True, q, r] // Simplify
```

Out[238]=

```
q || r
```

2. g (x, y) = ! (! (x + y) x + ! ! ! y) + xy + x! y

In[239]:=

```
g[x_, y_] := ! ((! (x ∨ y) ∧ x) ∨ ! ! ! y) ∨ (x ∧ y) ∨ (x ∧ ! y)
g[False, False]
```

Out[240]=

```
False
```

In[241]:=

```
g[False, True]
```

Out[241]=

```
True
```

3. h (x, y, z) = x (! (y + z)) + (xy + ! z) x

In[242]:=

```
h[x_, y_, z_] := (x ∧ (! (y ∨ z))) ∨ ((x ∧ y) ∨ ! z) ∧ x;
h[0, 0, 0] // Simplify
```

Out[243]=

```
False
```

In[244]:=

```
h[1, 0, 0] // Simplify
```

Out[244]=

```
True
```

In[245]:=

```
h[0, 1, 0] // Simplify
```

Out[245]=

```
False
```

In[246]:=

```
h[0, 0, 1] // Simplify
```

Out[246]=

```
False
```

In[247]:=

```
h[1, 1, 0] // Simplify
```

Out[247]=

```
True
```

In[248]:=

```
h[0, 1, 1] // Simplify
```

Out[248]=

```
False
```

In[249]:=

```
h[1, 0, 1] // Simplify
```

Out[249]=

```
False
```

Table Form

1. For Boolean Expression f :

In[250]:=

```
BooleanTable[{p, q, r, f[p, q, r]}, {p, q, r}] // TableForm
```

Out[250]//TableForm=

True	True	True	True
True	True	False	True
True	False	True	True
True	False	False	False
False	True	True	True
False	True	False	False
False	False	True	False
False	False	False	False

In[251]:=

```
Boole[BooleanTable[{p, q, r, f[p, q, r]}, {p, q, r}]] // TableForm
```

Out[251]//TableForm=

1	1	1	1
1	1	0	1
1	0	1	1
1	0	0	0
0	1	1	1
0	1	0	0
0	0	1	0
0	0	0	0

In[252]:=

```
TableForm[Boole[BooleanTable[{p, q, r, f[p, q, r]}, {p, q, r}]],
 TableHeadings → {None, {p, q, r, f}}]
```

Out[252]//TableForm=

p	q	r	f
1	1	1	1
1	1	0	1
1	0	1	1
1	0	0	0
0	1	1	1
0	1	0	0
0	0	1	0
0	0	0	0

2. For Boolean expression g:

In[253]:=

```
BooleanTable[{p, q, g[p, q]}, {p, q}] // TableForm
```

Out[253]//TableForm=

True	True	True
True	False	True
False	True	True
False	False	False

In[254]:=

```
Boole[BooleanTable[{p, q, g[p, q]}, {p, q}]] // TableForm
```

Out[254]//TableForm=

1	1	1
1	0	1
0	1	1
0	0	0

In[255]:=

```
TableForm[Boole[BooleanTable[{p, q, g[p, q]}, {p, q}]],
 TableHeadings → {None, {p, q, g}}]
```

Out[255]//TableForm=

p	q	g
1	1	1
1	0	1
0	1	1
0	0	0

3. For Boolean expression h :

In[256]:=

```
BooleanTable[{p, q, r, h[p, q, r]}, {p, q, r}] // TableForm
```

Out[256]//TableForm=

True	True	True	True
True	True	False	True
True	False	True	False
True	False	False	True
False	True	True	False
False	True	False	False
False	False	True	False
False	False	False	False

In[257]:=

```
Boole[BooleanTable[{p, q, r, h[p, q, r]}, {p, q, r}]] // TableForm
```

Out[257]//TableForm=

1	1	1	1
1	1	0	1
1	0	1	0
1	0	0	1
0	1	1	0
0	1	0	0
0	0	1	0
0	0	0	0

In[258]:=

```
TableForm[Boole[BooleanTable[{p, q, r, h[p, q, r]}, {p, q, r}]],
 TableHeadings → {None, {p, q, r, h}}]
```

Out[258]//TableForm=

p	q	r	h
1	1	1	1
1	1	0	1
1	0	1	0
1	0	0	1
0	1	1	0
0	1	0	0
0	0	1	0
0	0	0	0

Finding the following:1. Dual of a Boolean Polynomial/expression.2. Whether or not two Boolean polynomials are equivalent.3. Disjunctive normal form(Conjunctive Normal Form)from a given Boolean expression.4. Disjunctive normal form(Conjunctive Normal Form) when the Boolean polynomial expressed by a table of values.

1. Dual of a Boolean Polynomial/expression .

In[259]:=

```
f[x_, y_, z_] := (x ∧ y) ∨ (y ∧ z) ∨ (z ∧ x)
dual[exp_] := exp /. {And → Or, Or → And, Ture → False, False → True}
d[x_, y_, z_] = dual[f[x, y, z]]
```

Out[261]=

(x || y) && (y || z) && (z || x)

In[262]:=

```
% // Simplify
```

Out[262]=

(x && (y || z)) || (y && z)

In[263]:=

```
TableForm[Boole[BooleanTable[{p, q, r, d[p, q, r], f[p, q, r]}, {p, q, r}]],
 TableHeadings → {None, {p, q, r, d, f}}]
```

Out[263]//TableForm=

p	q	r	d	f
1	1	1	1	1
1	1	0	1	1
1	0	1	1	1
1	0	0	0	0
0	1	1	1	1
0	1	0	0	0
0	0	1	0	0
0	0	0	0	0

Ques. Representing a given circuit diagram (expressed using gates) in the form of Boolean expressions.To give Boolean expression of the given circuit diagram , we define each gate separately , and finally ask for the value of output G1.

In[265]:=

```
G4 = ¬ a;
```

In[266]:=

```
G5 = ¬ c;
```

In[267]:=

```
G2 = a ∧ b ;
```

In[268]:=

```
G3 = G4 ∧ G5 ∧ b ;
```

In[269]:=

```
G1 = G2 ∨ G3;
```

In[270]:=

```
G1
```

Out[270]=

(a && b) || (! a && ! c && b)

2. Gate Diagram

In[271]:=

```
G2 = a ∧ b ∧ c ;
```

In[272]:=

```
G4 = ¬ c;
```

In[273]:=

```
G3 = a ∧ b ∧ G4 ;
```

In[274]:=

```
G5 = ¬ a;
```

In[275]:=

```
G6 = ¬ c;
```

In[276]:=

```
G7 = G5 ∧ G6 ∧ b ;
```

In[277]:=

```
G1 = G2 ∨ G3 ∨ G7 ;
```

In[278]:=

```
G1
```

Out[278]=

```
(a && b && c) || (a && b && ! c) || (! a && ! c && b)
```

Ques . Minimize a given Boolean expression to find minimal expressions . To obtain minimised expressions in DNF , we can use booleanMinimize.

In[279]:=

```
BooleanMinimize[(a ∧ b) ∨ (! a ∧ b ∧ ! c)]
```

Out[279]=

```
(a && b) || (b && ! c)
```

To obtain minimized expression in CNF , we can use BooleanMinimize with specification for CNF as under

In[280]:=

```
BooleanMinimize[(a ∧ b) ∨ (! a ∧ b ∧ ! c), "CNF"]
```

Out[280]=

(a || ! c) && b

To obtain minimized expression in whichever form as minimum length , we can use simplify

In[281]:=

```
Simplify[(a ∧ b) ∨ (! a ∧ b ∧ ! c)]
```

Out[281]=

(a || ! c) && b