

UNIT-IV: BACKTRACKING, BRANCH AND BOUNDS→ Backtracking

General Method:

Let  $a_1, a_2, a_3, \dots, a_n$  be a set of 'n' tuples.

Now, first construct a partial solution vector from the given set.

 $\{a_1, \dots, a_{1k}, a_{1k+1}\} \Rightarrow$  Partial Solution Vector

First, take one tuple, and then add other, then check for the constraints of the problem, if that satisfies, add the other tuple, else, go to first tuple (backtracking) and change it.

Problems in Backtracking:

i) 8-Queens Problem

ii) Graph Colouring

iii) Hamilton Cycles

iv) 0/1 Knapsack

→ 8-Queens Problem:

→ 4 Queens

Solution Vector:

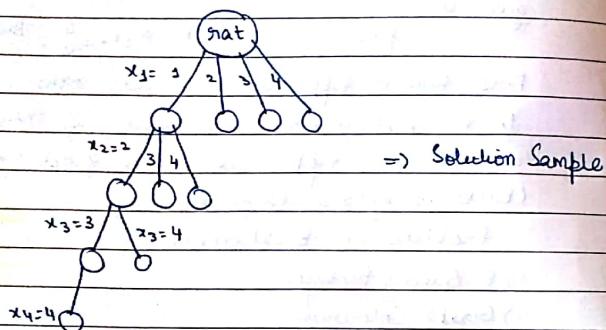
$$\left\{ \begin{array}{c} x_1 \\ x_2 \\ x_3 \\ x_4 \end{array} \right\}$$

x	1	3	2
✓	2	4	1

	Q <sub>1</sub>		
		Q <sub>2</sub>	
			Q <sub>3</sub>
			Q <sub>4</sub>

Problem constraint: No two queens should be attacked  
There are two constraints in this problem:

- Explicit Constraint  $\Rightarrow x_i \leq n$  and  $x_i = 0$  or 1
- Implicit Constraint  $\Rightarrow$  To set some solution vector from solution sample



Out of the solution sample, we have to select the solution vector which satisfies all the constraints of the problem.

#### General Method of Backtracking (Recursive Algorithm)

Algorithm BackTrack ( $k$ )

```

1 for each  $x[k] \in T (x[1], x[2], \dots, x[k-1])$  do
  2 if ( $B \times (x[i], \dots, x[k]) \neq 0$ ) then
    3   if ( $(x[i], \dots, x[k])$  is a path to answer node
        then write  $x[i-k]$ 
    4   if ( $k < n$ ) then BackTrack ( $k+1$ )
  5
  6
  7
  8
  9
  10
  11
  12
  13
  14
  15
  16
  17
  18
  19
  20
  21
  22
  23
  24
  25
  26
  27
  28
  29
  30
  31
  32
  33
  34
  35
  36
  37
  38
  39
  40
  41
  42
  43
  44
  45
  46
  47
  48
  49
  50
  51
  52
  53
  54
  55
  56
  57
  58
  59
  60
  61
  62
  63
  64
  65
  66
  67
  68
  69
  70
  71
  72
  73
  74
  75
  76
  77
  78
  79
  80
  81
  82
  83
  84
  85
  86
  87
  88
  89
  90
  91
  92
  93
  94
  95
  96
  97
  98
  99
  100
  101
  102
  103
  104
  105
  106
  107
  108
  109
  110
  111
  112
  113
  114
  115
  116
  117
  118
  119
  120
  121
  122
  123
  124
  125
  126
  127
  128
  129
  130
  131
  132
  133
  134
  135
  136
  137
  138
  139
  140
  141
  142
  143
  144
  145
  146
  147
  148
  149
  150
  151
  152
  153
  154
  155
  156
  157
  158
  159
  160
  161
  162
  163
  164
  165
  166
  167
  168
  169
  170
  171
  172
  173
  174
  175
  176
  177
  178
  179
  180
  181
  182
  183
  184
  185
  186
  187
  188
  189
  190
  191
  192
  193
  194
  195
  196
  197
  198
  199
  200
  201
  202
  203
  204
  205
  206
  207
  208
  209
  210
  211
  212
  213
  214
  215
  216
  217
  218
  219
  220
  221
  222
  223
  224
  225
  226
  227
  228
  229
  230
  231
  232
  233
  234
  235
  236
  237
  238
  239
  240
  241
  242
  243
  244
  245
  246
  247
  248
  249
  250
  251
  252
  253
  254
  255
  256
  257
  258
  259
  260
  261
  262
  263
  264
  265
  266
  267
  268
  269
  270
  271
  272
  273
  274
  275
  276
  277
  278
  279
  280
  281
  282
  283
  284
  285
  286
  287
  288
  289
  290
  291
  292
  293
  294
  295
  296
  297
  298
  299
  300
  301
  302
  303
  304
  305
  306
  307
  308
  309
  310
  311
  312
  313
  314
  315
  316
  317
  318
  319
  320
  321
  322
  323
  324
  325
  326
  327
  328
  329
  330
  331
  332
  333
  334
  335
  336
  337
  338
  339
  340
  341
  342
  343
  344
  345
  346
  347
  348
  349
  350
  351
  352
  353
  354
  355
  356
  357
  358
  359
  360
  361
  362
  363
  364
  365
  366
  367
  368
  369
  370
  371
  372
  373
  374
  375
  376
  377
  378
  379
  380
  381
  382
  383
  384
  385
  386
  387
  388
  389
  390
  391
  392
  393
  394
  395
  396
  397
  398
  399
  400
  401
  402
  403
  404
  405
  406
  407
  408
  409
  410
  411
  412
  413
  414
  415
  416
  417
  418
  419
  420
  421
  422
  423
  424
  425
  426
  427
  428
  429
  430
  431
  432
  433
  434
  435
  436
  437
  438
  439
  440
  441
  442
  443
  444
  445
  446
  447
  448
  449
  450
  451
  452
  453
  454
  455
  456
  457
  458
  459
  460
  461
  462
  463
  464
  465
  466
  467
  468
  469
  470
  471
  472
  473
  474
  475
  476
  477
  478
  479
  480
  481
  482
  483
  484
  485
  486
  487
  488
  489
  490
  491
  492
  493
  494
  495
  496
  497
  498
  499
  500
  501
  502
  503
  504
  505
  506
  507
  508
  509
  510
  511
  512
  513
  514
  515
  516
  517
  518
  519
  520
  521
  522
  523
  524
  525
  526
  527
  528
  529
  530
  531
  532
  533
  534
  535
  536
  537
  538
  539
  540
  541
  542
  543
  544
  545
  546
  547
  548
  549
  550
  551
  552
  553
  554
  555
  556
  557
  558
  559
  560
  561
  562
  563
  564
  565
  566
  567
  568
  569
  570
  571
  572
  573
  574
  575
  576
  577
  578
  579
  580
  581
  582
  583
  584
  585
  586
  587
  588
  589
  590
  591
  592
  593
  594
  595
  596
  597
  598
  599
  600
  601
  602
  603
  604
  605
  606
  607
  608
  609
  610
  611
  612
  613
  614
  615
  616
  617
  618
  619
  620
  621
  622
  623
  624
  625
  626
  627
  628
  629
  630
  631
  632
  633
  634
  635
  636
  637
  638
  639
  640
  641
  642
  643
  644
  645
  646
  647
  648
  649
  650
  651
  652
  653
  654
  655
  656
  657
  658
  659
  660
  661
  662
  663
  664
  665
  666
  667
  668
  669
  670
  671
  672
  673
  674
  675
  676
  677
  678
  679
  680
  681
  682
  683
  684
  685
  686
  687
  688
  689
  690
  691
  692
  693
  694
  695
  696
  697
  698
  699
  700
  701
  702
  703
  704
  705
  706
  707
  708
  709
  710
  711
  712
  713
  714
  715
  716
  717
  718
  719
  720
  721
  722
  723
  724
  725
  726
  727
  728
  729
  730
  731
  732
  733
  734
  735
  736
  737
  738
  739
  740
  741
  742
  743
  744
  745
  746
  747
  748
  749
  750
  751
  752
  753
  754
  755
  756
  757
  758
  759
  760
  761
  762
  763
  764
  765
  766
  767
  768
  769
  770
  771
  772
  773
  774
  775
  776
  777
  778
  779
  780
  781
  782
  783
  784
  785
  786
  787
  788
  789
  790
  791
  792
  793
  794
  795
  796
  797
  798
  799
  800
  801
  802
  803
  804
  805
  806
  807
  808
  809
  810
  811
  812
  813
  814
  815
  816
  817
  818
  819
  820
  821
  822
  823
  824
  825
  826
  827
  828
  829
  830
  831
  832
  833
  834
  835
  836
  837
  838
  839
  840
  841
  842
  843
  844
  845
  846
  847
  848
  849
  850
  851
  852
  853
  854
  855
  856
  857
  858
  859
  860
  861
  862
  863
  864
  865
  866
  867
  868
  869
  870
  871
  872
  873
  874
  875
  876
  877
  878
  879
  880
  881
  882
  883
  884
  885
  886
  887
  888
  889
  890
  891
  892
  893
  894
  895
  896
  897
  898
  899
  900
  901
  902
  903
  904
  905
  906
  907
  908
  909
  910
  911
  912
  913
  914
  915
  916
  917
  918
  919
  920
  921
  922
  923
  924
  925
  926
  927
  928
  929
  930
  931
  932
  933
  934
  935
  936
  937
  938
  939
  940
  941
  942
  943
  944
  945
  946
  947
  948
  949
  950
  951
  952
  953
  954
  955
  956
  957
  958
  959
  960
  961
  962
  963
  964
  965
  966
  967
  968
  969
  970
  971
  972
  973
  974
  975
  976
  977
  978
  979
  980
  981
  982
  983
  984
  985
  986
  987
  988
  989
  990
  991
  992
  993
  994
  995
  996
  997
  998
  999
  1000
  1001
  1002
  1003
  1004
  1005
  1006
  1007
  1008
  1009
  1010
  1011
  1012
  1013
  1014
  1015
  1016
  1017
  1018
  1019
  1020
  1021
  1022
  1023
  1024
  1025
  1026
  1027
  1028
  1029
  1030
  1031
  1032
  1033
  1034
  1035
  1036
  1037
  1038
  1039
  1040
  1041
  1042
  1043
  1044
  1045
  1046
  1047
  1048
  1049
  1050
  1051
  1052
  1053
  1054
  1055
  1056
  1057
  1058
  1059
  1060
  1061
  1062
  1063
  1064
  1065
  1066
  1067
  1068
  1069
  1070
  1071
  1072
  1073
  1074
  1075
  1076
  1077
  1078
  1079
  1080
  1081
  1082
  1083
  1084
  1085
  1086
  1087
  1088
  1089
  1090
  1091
  1092
  1093
  1094
  1095
  1096
  1097
  1098
  1099
  1100
  1101
  1102
  1103
  1104
  1105
  1106
  1107
  1108
  1109
  1110
  1111
  1112
  1113
  1114
  1115
  1116
  1117
  1118
  1119
  1120
  1121
  1122
  1123
  1124
  1125
  1126
  1127
  1128
  1129
  1130
  1131
  1132
  1133
  1134
  1135
  1136
  1137
  1138
  1139
  1140
  1141
  1142
  1143
  1144
  1145
  1146
  1147
  1148
  1149
  1150
  1151
  1152
  1153
  1154
  1155
  1156
  1157
  1158
  1159
  1160
  1161
  1162
  1163
  1164
  1165
  1166
  1167
  1168
  1169
  1170
  1171
  1172
  1173
  1174
  1175
  1176
  1177
  1178
  1179
  1180
  1181
  1182
  1183
  1184
  1185
  1186
  1187
  1188
  1189
  1190
  1191
  1192
  1193
  1194
  1195
  1196
  1197
  1198
  1199
  1200
  1201
  1202
  1203
  1204
  1205
  1206
  1207
  1208
  1209
  1210
  1211
  1212
  1213
  1214
  1215
  1216
  1217
  1218
  1219
  1220
  1221
  1222
  1223
  1224
  1225
  1226
  1227
  1228
  1229
  1230
  1231
  1232
  1233
  1234
  1235
  1236
  1237
  1238
  1239
  1240
  1241
  1242
  1243
  1244
  1245
  1246
  1247
  1248
  1249
  1250
  1251
  1252
  1253
  1254
  1255
  1256
  1257
  1258
  1259
  1260
  1261
  1262
  1263
  1264
  1265
  1266
  1267
  1268
  1269
  1270
  1271
  1272
  1273
  1274
  1275
  1276
  1277
  1278
  1279
  1280
  1281
  1282
  1283
  1284
  1285
  1286
  1287
  1288
  1289
  1290
  1291
  1292
  1293
  1294
  1295
  1296
  1297
  1298
  1299
  1300
  1301
  1302
  1303
  1304
  1305
  1306
  1307
  1308
  1309
  1310
  1311
  1312
  1313
  1314
  1315
  1316
  1317
  1318
  1319
  1320
  1321
  1322
  1323
  1324
  1325
  1326
  1327
  1328
  1329
  1330
  1331
  1332
  1333
  1334
  1335
  1336
  1337
  1338
  1339
  1340
  1341
  1342
  1343
  1344
  1345
  1346
  1347
  1348
  1349
  1350
  1351
  1352
  1353
  1354
  1355
  1356
  1357
  1358
  1359
  1360
  1361
  1362
  1363
  1364
  1365
  1366
  1367
  1368
  1369
  1370
  1371
  1372
  1373
  1374
  1375
  1376
  1377
  1378
  1379
  1380
  1381
  1382
  1383
  1384
  1385
  1386
  1387
  1388
  1389
  1390
  1391
  1392
  1393
  1394
  1395
  1396
  1397
  1398
  1399
  1400
  1401
  1402
  1403
  1404
  1405
  1406
  1407
  1408
  1409
  1410
  1411
  1412
  1413
  1414
  1415
  1416
  1417
  1418
  1419
  1420
  1421
  1422
  1423
  1424
  1425
  1426
  1427
  1428
  1429
  1430
  1431
  1432
  1433
  1434
  1435
  1436
  1437
  1438
  1439
  1440
  1441
  1442
  1443
  1444
  1445
  1446
  1447
  1448
  1449
  1450
  1451
  1452
  1453
  1454
  1455
  1456
  1457
  1458
  1459
  1460
  1461
  1462
  1463
  1464
  1465
  1466
  1467
  1468
  1469
  1470
  1471
  1472
  1473
  1474
  1475
  1476
  1477
  1478
  1479
  1480
  1481
  1482
  1483
  1484
  1485
  1486
  1487
  1488
  1489
  1490
  1491
  1492
  1493
  1494
  1495
  1496
  1497
  1498
  1499
  1500
  1501
  1502
  1503
  1504
  1505
  1506
  1507
  1508
  1509
  1510
  1511
  1512
  1513
  1514
  1515
  1516
  1517
  1518
  1519
  1520
  1521
  1522
  1523
  1524
  1525
  1526
  1527
  1528
  1529
  1530
  1531
  1532
  1533
  1534
  1535
  1536
  1537
  1538
  1539
  1540
  1541
  1542
  1543
  1544
  1545
  1546
  1547
  1548
  1549
  1550
  1551
  1552
  1553
  1554
  1555
  1556
  1557
  1558
  1559
  1560
  1561
  1562
  1563
  1564
  1565
  1566
  1567
  1568
  1569
  1570
  1571
  1572
  1573
  1574
  1575
  1576
  1577
  1578
  1579
  1580
  1581
  1582
  1583
  1584
  1585
  1586
  1587
  1588
  1589
  1590
  1591
  1592
  1593
  1594
  1595
  1596
  1597
  1598
  1599
  1600
  1601
  1602
  1603
  1604
  1605
  1606
  1607
  1608
  1609
  1610
  1611
  1612
  1613
  1614
  1615
  1616
  1617
  1618
  1619
  1620
  1621
  1622
  1623
  1624
  1625
  1626
  1627
  1628
  1629
  1630
  1631
  1632
  1633
  1634
  1635
  1636
  1637
  1638
  1639
  1640
  1641
  1642
  1643
  1644
  1645
  1646
  1647
  1648
  1649
  1650
  1651
  1652
  1653
  1654
  1655
  1656
  1657
  1658
  1659
  1660
  1661
  1662
  1663
 
```

### Algorithm of N-Queens

Algorithm NQueens ( $k, n$ )

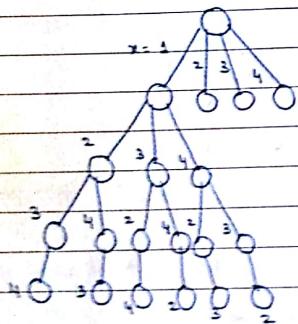
```

for i := 1 to n do
    if (place ( $x, i$ )) then
         $x[k] := i$ ;
        if ( $k = n$ ) then write ( $x[1], \dots, n$ )
        else NQueens ( $k+1, n$ )
    }
}

```

}

### Possible Solutions to Select Solution Vector for 4-Queens



Total number of possible solutions = 24

### 8-Queens Solution:

q1									
									q2
									q3
									q4
									q5
									q6
									q7
									q8
									q9
									q10

### Partial Solution Vector:

$\{x_1 \ x_2 \ x_3 \ x_4 \ x_5 \ x_6 \ x_7 \ x_8\}$   
 1    2    5    8    2    4    6    3

P.T.O



## Graph Colouring

General Method:

Construct partial solution vector  $x_1, x_2, x_3, \dots, x_n$ .  
If the new element  $x_{n+1}$  satisfies problem constraints, add it to solution vector, else, apply backtracking.

Problem Statement:

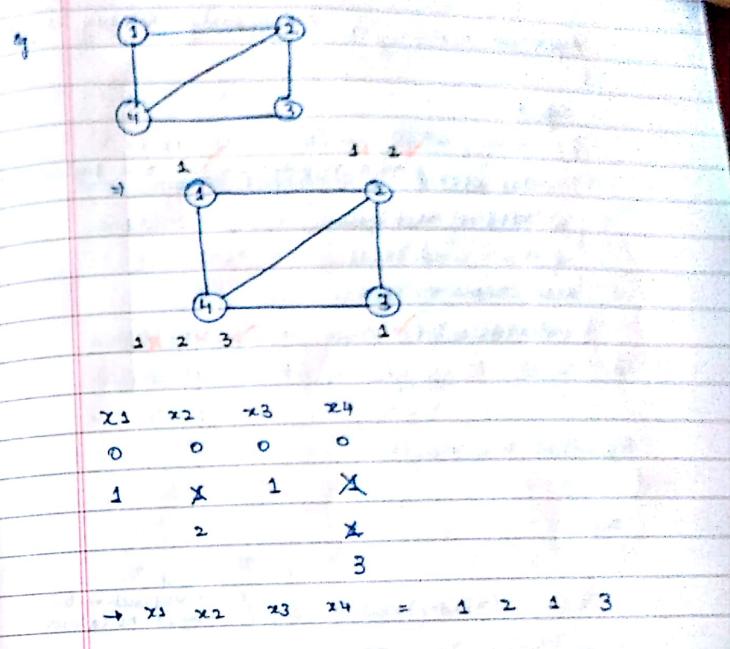
Given undirected graph 'G' has 'n' vertices, we must assign 'm' number of colours (minimum no. of colours) so that no two vertices have same colour.

$x_1$	$x_2$	$x_3$	$\dots$	$x_n$
0	0	0	0	= Step 1: Assign no colour to all vertices
1	2	3	4	= Step 2: Assign adjacent colours to nodes
1	2	3	4	= Step 3: Assign colour 2 to node 1 because node 1 is adjacent to node 2 and node 2 has the same colour as colour of node 1, apply colour 2 to node 2. Continue till $x_{n+1}$

Construct Binary Adjacency Matrix for given graph 'G'.

$G(i,j) = 1 \Rightarrow i$  and  $j$  are adjacent nodes

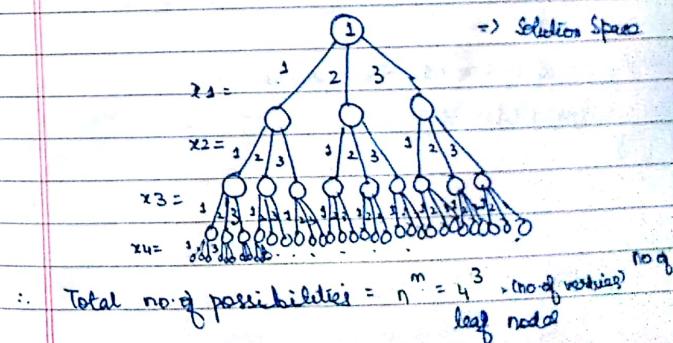
$G(i,j) = 0 \Rightarrow i$  and  $j$  are not adjacent nodes



State Space / Solution space for the graph

= 4 levels else

=  $x_1 x_2 x_3 x_4$



$\therefore$  Total no. of possibilities =  $n^m = 4^3$   $\rightarrow$  (no. of vertices)<sup>No. of levels</sup>

Algorithm mcoloring(k) // Assume partial solution is given up to k-1

{ repeat

{

    nextcolor(k) // return the colour assigned to a particular node  
        if ( $x[v] = 0$ ) then return;  
        if ( $k = n$ ) write ( $x[1], \dots, x[n]$ )  
        else mcoloring ( $k+1$ );  
    } until (false)

}

Algorithm nextcolor(k)

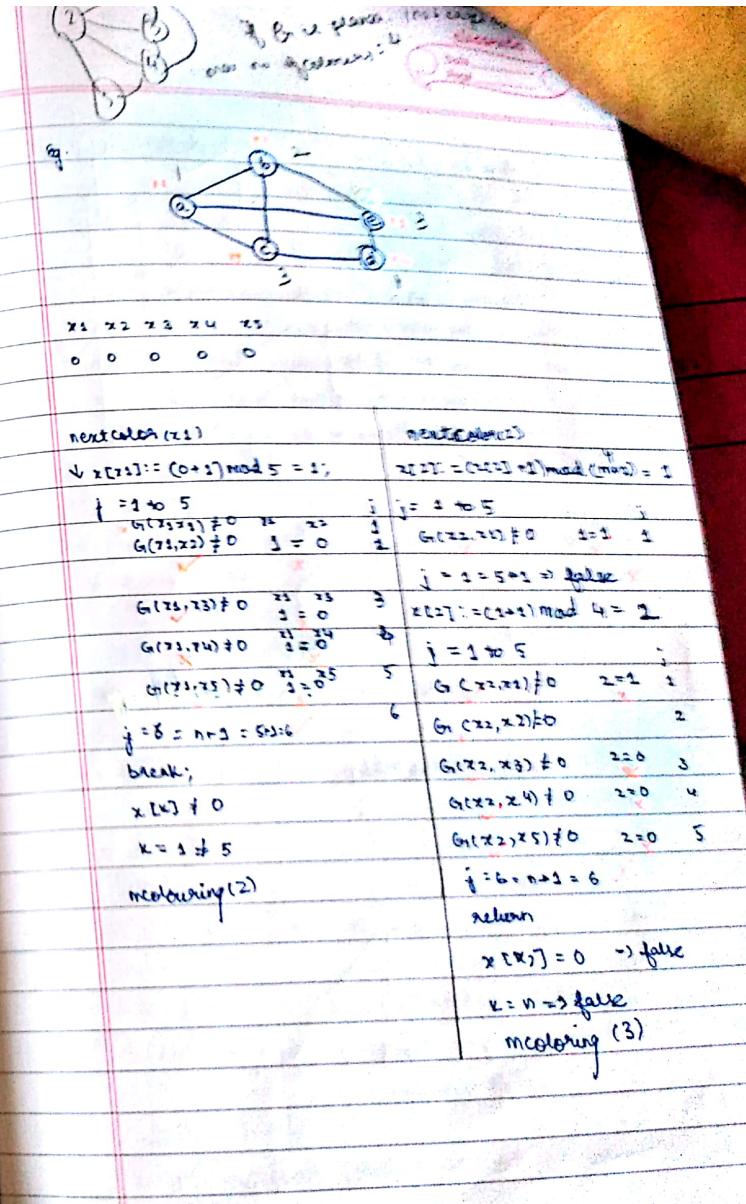
{

repeat

{

$x[v] := (x[v]+1) \bmod (m-1)$ ; // initially each vertex is  
        if ( $x[v] = 0$ ) then return;  
        for  $j := 1$  to  $n$  do  
            {  
                 $\text{(adjacency condition)} \quad \text{(completeness condition)}$   
                if ( $G(v, j) \neq 0$  and  $x[v] = x[j]$ )  
                    break;  
            }  
            if ( $j = n+1$ ) then return;  
        } until (false)

}



→ Hamiltonian Cycle

Problem Statement:

Given Graph  $G_i$  travels all edges exactly once and reaches to starting point. The path contains  $(n+1)$  vertices,  $v_1$  and  $v_{n+1}$  are the same. Start at vertex 1, visit all other edges vertices by travelling all edges exactly once and reach back to starting point.

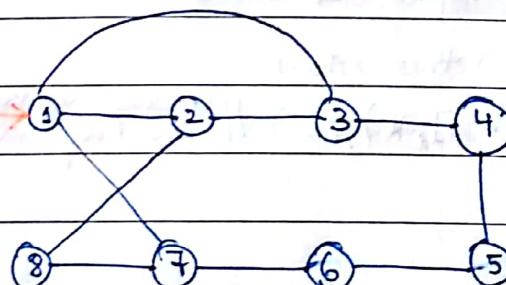
$$x_1 \quad x_2 \quad x_3 \dots \dots \dots \quad x_n \quad x_{n+1} \rightarrow \text{Initial solution vector}$$

To select new vertex,  $k^{th}$  vertex, we have to verify 2 conditions:

- i)  $x_{k-1}$  and  $x_k$ , both vertices are adjacent
- ii) Path is generated upto  $k-1^{th}$  vertex

To select last vertex, we must verify if a path exists from  $x_n$  to  $x_{n-1}$

e.g.



2 paths:

(i) 1, 3, 4, 5, 6, 7, 8, 2, 1

(ii) 1, 2, 8, 7, 6, 5, 4, 3, 1

NOTE: It is like travelling salesman problem, but we consider different weights in travelling Salesman problem, but in this problem, all the vertices have same distance, i.e., all the edges have same values.

Binary Adjacency Matrix

$G(i,j)=1 \Rightarrow i, j$  are adjacent       $G(i,j)=0 \Rightarrow i, j$  are not adjacent

### Algorithm hamiltonian (x)

```
repeat
    next vertex (k); // returns highest index adjacent to k
    if ( $x[k] = 0$ ) then return
    if ( $k = n$ ) then write( $x[1] \dots x[n]$ ) // write solution vector
    else
        Hamiltonian ( $x+1$ )
    until (false)
```

### Algorithm nextVertex (x)

```
repeat
     $x[k] = (x[k] + 1) \bmod (n+1)$ ;
    if ( $x[k] = 0$ ) then return
    if ( $G(x[k-1], x[k]) \neq 0$ ) // checking if vertices k-1 and k
    then
        for  $j := 1$  to  $k-1$  // to verify if the vertex  $x[j]$  is distinct
        from all other vertices
        if ( $x[j] = x[k]$ ) then break; // vertex is visited
        if ( $j = k$ )
            if ( $(k < n)$  or ( $k = n$ ) and  $G(x[n], x[1]) \neq 0$ )
                then return;
```

### Branch and Bound

#### General Method:

We find solution vector from solution space tree.

We have three types of branching functions to construct state space tree.

i) FIFO - First In First Out

ii) LIFO - Last In First Out

iii) LC-Search - least Cause Function

We need to discuss about 3 nodes in solution space tree or to construct state space tree.

i) Live Node (L node)

- The node is generated but not explored.

ii) Explore Node (E node)

- The node is generated and being explored currently.

iii) Dead Node

- The node cannot generate solution vector.

#### Difference between Branch and Bound and Backtracking

Branch and Bound contains branching function, we use it to generate solution / state space tree, then using bounding function, we decide whether we can explore the tree or not, but no such functions exist in Backtracking Method.

While generating solution space tree, we generate a bounding function and use it to explore the current node.

### Live Node

- Live Node is a node that has been generated, but both children have not yet been generated.

### Explore Node

- Explore Node is a live node whose children are currently being explored.

### Dead Node

- Dead Node is a generated node that is not to be expanded or explored any further.

We have 3 problems in Branch and Bound

→ 15 Puzzle Problem

→ Travelling Salesman Problem [TSP]

→ 0/1 Knapsack Problem

### 15 Puzzle Problem

#### Problem Statement

- We take one frame, the frame is divided into 16 slots, each slot is assigned one number, the position is empty, we perform some moves so that the final state/goal state is reached.

1 <sub>1</sub>	3 <sub>2</sub>	4 <sub>3</sub>	15 <sub>4</sub>
2 <sub>5</sub>	5 <sub>6</sub>	12 <sub>7</sub>	
7 <sub>9</sub>	6 <sub>10</sub>	14 <sub>11</sub>	
8 <sub>13</sub>	9 <sub>14</sub>	10 <sub>15</sub>	13 <sub>16</sub>

Initial State

1 <sub>1</sub>	2 <sub>2</sub>	3 <sub>3</sub>	4 <sub>4</sub>
5 <sub>5</sub>	6 <sub>6</sub>	7 <sub>7</sub>	8 <sub>8</sub>
9 <sub>9</sub>	10 <sub>10</sub>	11 <sub>11</sub>	12 <sub>12</sub>
13 <sub>13</sub>	14 <sub>14</sub>	15 <sub>15</sub>	16 <sub>16</sub>

Goal State



By using Brute Force, it takes  $16!$  possibilities, which is very difficult to search.

So, we use Branch & Bound.

We apply LC-search bound function.

For each and every final node, we must calculate:

$$C(x) = f(h(x)) + g(x)$$

where  $x = \text{node}$ ,  $f(x) = \text{function (non-decreasing fn)}$

$h(x) = \text{distance from root node to node } x$

$g(x) = \text{Any extra effort we are putting to reach root node to node } x'$

$C(x) = \text{cost function}$

Position of  $i$  = slot numbers (in black) in given frame

For given initial state, we find whether it is possible to reach goal state or not by using the formula:

$$\sum_{i=1}^{16} \text{less}(i) + x$$

→ If the function returns even value, then the solution is possible

→ If the function returns odd value, then the solution is not possible

→  $\text{less}(i)$  is no. of times  $j$  such that  $j$  is less than  $i$

but position of  $i >$  position of  $j$

Eg. 

1	3	2
---	---	---

$\text{less}(1) = 0 \Rightarrow$  no element after 1 is less than 1

$\text{less}(3) = 1 \Rightarrow$  1 element after 3 is less than 3

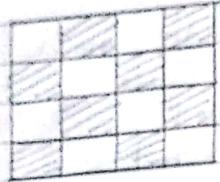
$\text{less}(2) = 0 \Rightarrow$  no element after 2 is less than 2

Eg.	1	3	4	15	$\text{less}(1) = 0$	$\text{less}(2) = 0$	$\text{less}(3) = 1$
	2		5	12	$\text{less}(2) = 1$	$\text{less}(5) = 10$	$\text{less}(12) = 0$
	7	6	11	14	$\text{less}(7) = 1$	$\text{less}(6) = 10$	$\text{less}(11) = 3$
	8	9	10	13	$\text{less}(8) = 11$	$\text{less}(9) = 10$	$\text{less}(10) = 6$
					$\text{less}(13) = 11$	$\text{less}(10) = 6$	$\text{less}(13) = 4$

Consider empty cell value = 16

$\sum \text{less}(i) + \frac{1}{2} \times 16 = \frac{\sum \text{less}(i)}{2} + 8 = 37$   $x = 0$   $\text{less}(8) = 0$   $\text{less}(9) = 0$   $\text{less}(10) = 0$   $\text{less}(11) = 3$

$\Rightarrow \sum \text{less}(i) + \frac{1}{2} \times 16 = \frac{\sum \text{less}(i)}{2} + 8 = 37$   $\Rightarrow$  total is possible



Shaded Region:

$x$  value = 3

Non shaded Region:

$x$  value = 0

4.	1	2	3	4
	108(1)=0	108(5)=0	108(9)=1	108(13)=1
5	6	8	108(2)=0	108(6)=0
9	10	7	108(3)=0	108(7)=9
13	14	15	108(4)=0	108(8)=1

$$\sum_{i=1}^{15} 108(i) = 15 \quad x = \text{shaded region} = 1$$

$$\therefore \sum_{i=1}^{15} 108(i) + x = 15 + 1 = 16 = \text{even} \Rightarrow \text{solution is possible}$$

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

left right down up

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

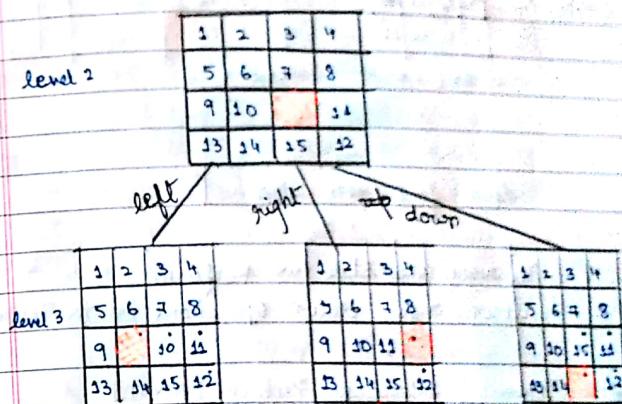
Bound function for left =  $1+5+6$

Bound function for right =  $1+6+6$

Bound function for down =  $1+3+4$

Bound function for up =  $1+3+6$

Which node gives min cost = currently exploring node  
Perform left, right, down op on min cost node,  
exclude opposite operation (up)



Bound function for left =  $2+4 = 6$

Bound function for right =  $2+2 = 4$

Bound function for down =  $2+4 = 6$

Which node gives min cost = currently exploring node  
Perform left, right, left, down op on min cost node,  
exclude opposite operation (down (left))

Calculate bounding function for each and every node

$$C(x) = f(h(x)) + g(x)$$

$h(x)$  = difference in levels

$g(x)$  = no. of numbers different from Node in levels to goal state

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

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

up      down      left right  
cost per step

$c = 3 + 3 = 6$        $c = 3 + 0 = 3$

down = final / goal state

∴ To reach goal state, we perform down, right, down operations on the initial state.

#### → Travelling Salesperson Problem [TSP]

Problem Statement : We must cover all the vertices in the given graph (directed) with minimum cost, exactly once and reaches back to same original city.

$G(V, E) \Rightarrow$  Directed Graph (Cities, Edges)

$(1, n) \Rightarrow$  Starts at vertex 1, covers all other cities  $(n-1) \times n$  exactly once and returns back to vertex 1

Using Brute Force, Time Complexity =  $(n-1)! = n!$

By using LC-Search, we find a solution for the problem

Selection has two steps

Step 2 : Find out reduced cost matrix

↳ a) Find Row Reduced Cost Matrix (R)

(Identify minimum element from each row in cost matrix, subtract minimum element from each element in the row)

↳ b) Find Column Reduced Cost Matrix

(Identify minimum element from each column in new reduced cost matrix, subtract minimum element from each element in the column)

↳ c) Find Cumulative reduced cost matrix

(row cost + column cost)

Step 2 :  $\hat{C}(S) = C(R) + A(i,j) + s$

↳ a)  $(i,j) \Rightarrow$  Change all entries of row i in matrix A to infinite

↳ b)  $(i,j) \Rightarrow$  Change all entries of column j in matrix A to infinite

↳ c)  $(i,j) \Rightarrow$  Change  $C_{ij}$  entry to infinite

R = Root node    A[] = Reduced cost matrix

S = minimum cost child node , S be a child of R, not leaf

$C(S)$  = Cost of child node S     $C(R)$  = Reduced cost of matrix A

$A(i,j) = A(R,S)$      $s$  = Reduced cost of S

e.g. Given cost Malawi

	1	2	3	4	5
1	∞	20	30	10	11
2	15	∞	36	4	2
3	3	5	∞	2	4
4	19	6	18	∞	3
5	26	4	7	16	∞

## Row Reduced Cost Matrix

	1	2	3	4	5	
1	∞	10	20	0	1	
2	13	∞	14	2	0	2
3	1	3	∞	0	2	2
4	16	3	45	∞	0	3
5	12	0	3	12	∞	4

### Column Reduced Cost Matrix

	1	2	3	4	5
1	00	10	17	0	1
2	12	00	18	2	0
3	0	3	00	0	2
4	15	3	12	00	0
5	11	0	0	13	00

$$\therefore \text{Cost}(R) = c(R) = 9\text{cost} + 6\text{cost} = 21 + 4 = 25$$

1 0 3 0 0  
→ Column Redu...

$$\text{Cost} = 2+6+3+0+0 \\ = 11$$

11

$$\hat{C}(s) = C(s) + A(i,j) + \beta_1 g$$

$$\hat{C}(g) = C(R) + A(1,2) + g$$

$$= 25 + 10 + 0 = 35$$

9:	1	2	3	4	5	
3	00	00	00	00	00	- 0
2	00	00	11	2	0	- 0
3	02	00	00	0	2	- 0
4	05	00	12	00	0	- 0
5	13	00	0	12	00	- 0

$$g_1 = a \cos t + c \cos t = 0 + 0 = 0$$

$$\hat{C}(s) = C(R) + A(i,j) + g$$

$$= 25 + 17 + 11 = 53$$

9:	1	2	3	4	5	
1	00	00	00	00	00	-0
2	12	00	00	2	0	-0
3	00	3	00	0	2	-0
4	15	3	00	00	0	-0
5	11	0	00	12	00	-0
	13	0	0	0	0	

$$g = g_{\text{west}} + c_{\text{west}} = 0 + 11 = 11$$

$$\hat{c}(5) = c(cR) + A(5, j) + g_1$$

$$\hat{c}(4) = c(cR) + A(4, 4) + g_1$$

$$= 25 + 0 + 0 = 25$$

Currently exploring node = minimum cost node = 3 to 4

$g_i$	1	2	3	4	5	
1	0	0	0	0	0	-0
2	12	0	11	0	0	-0
3	0	3	0	0	2	-0
4	0	3	12	0	0	-0
5	11	0	0	0	0	-0
	0	1	0	0	0	

$$g_i: g_{cost} + c_{cost} = 0 + 0 = 0$$

$$\hat{c}(5) = c(cR) + A(5, j) + g_1$$

$$\hat{c}(5) = c(cR) + A(5, 5) + g_1$$

$$= 25 + 1 + 5 = 31$$

$$g_i: 1 \quad 2 \quad 3 \quad 4 \quad 5$$

$g_i$	1	2	3	4	5	
1	0	0	0	0	0	-0
2	12	0	11	0	0	-0
3	0	3	0	0	2	-0
4	0	3	12	0	0	-0
5	11	0	0	0	0	-0
	0	1	0	0	0	

$$\hat{c}(2) = c(cR) + A(4, 2) + g_1 \quad \hat{c}(3) = c(cR) + A(4, 3) + g_1$$

$$= 25 + 0 + 0 = 25 \quad = 25 + 0 + 0 = 25$$

$$g_i: 1 \quad 2 \quad 3 \quad 4 \quad 5$$

$g_i$	1	2	3	4	5	
1	0	0	0	0	0	-0
2	12	0	11	0	0	-0
3	0	3	0	0	2	-0
4	0	3	12	0	0	-0
5	11	0	0	0	0	-0
	0	1	0	0	0	

$$\hat{c}(5) = c(cR) + A(4, 5) + g_1 = 25 + 0 + 1 = 26$$

$$g_i: 1 \quad 2 \quad 3 \quad 4 \quad 5$$

$g_i$	1	2	3	4	5	
1	0	0	0	0	0	-0
2	10	0	9	0	0	-1
3	0	3	0	0	0	-0
4	12	0	9	0	0	-0
5	11	0	0	0	0	-0
	0	1	0	0	0	

node = minimum cost node  
= 10

$g_i$	1	2	3	4	5	
1	0	0	0	0	0	-0
2	10	0	9	0	0	-0
3	0	3	0	0	0	-0
4	12	0	9	0	0	-0
5	10	0	0	12	0	-0
	0	0	0	0	0	

$$g_i: g_{cost} + c_{cost} = 10 + 0 = 10$$

$$= 10$$

$g_i$	1	2	3	4	5	
1	0	0	0	0	0	-0
2	12	0	11	0	0	-1
3	0	3	0	0	0	-0
4	8	0	0	0	0	-0
5	11	0	0	0	0	-0
	0	1	0	0	0	

$$g_i: g_{cost} + c_{cost} = 8 + 0 = 8$$

$$= 8$$

$g_i$	1	2	3	4	5	
1	0	0	0	0	0	-0
2	12	0	11	0	0	-1
3	0	3	0	0	0	-0
4	12	0	9	0	0	-0
5	11	0	0	0	0	-0
	0	1	0	0	0	

$$g_i: g_{cost} + c_{cost} = 12 + 0 = 12$$

$$= 12$$

$g_i$	1	2	3	4	5	
1	0	0	0	0	0	-0
2	12	0	11	0	0	-1
3	0	3	0	0	0	-0
4	12	0	9	0	0	-0
5	11	0	0	0	0	-0
	0	1	0	0	0	

$$g_i: g_{cost} + c_{cost} = 11 + 0 = 11$$

$$= 11$$

$g_i$	1	2	3	4	5	
1	0	0	0	0	0	-0
2	12	0	11	0	0	-1
3	0	3	0	0	0	-0
4	8	0	0	0	0	-0
5	11	0	0	0	0	-0
	0	1	0	0	0	

$$g_i: g_{cost} + c_{cost} = 8 + 0 = 8$$

$$= 8$$

$g_i$	1	2	3	4	5	
1	0	0	0	0	0	-0
2	12	0	11	0	0	-1
3	0	3	0	0	0	-0
4	12	0	9	0	0	-0
5	11	0	0	0	0	-0
	0	1	0	0	0	

$$g_i: g_{cost} + c_{cost} = 12 + 0 = 12$$

$$= 12$$

$g_i$	1	2	3	4	5	
1	0	0	0	0	0	-0
2	12	0	11	0	0	-1
3	0	3	0	0	0	-0
4	12	0	9	0	0	-0
5	11	0	0	0	0	-0
	0	1	0	0	0	

$$g_i: g_{cost} + c_{cost} = 11 + 0 = 11$$

$$= 11$$

$g_i$	1	2	3	4	5	
1	0	0	0	0	0	-0
2	12	0	11	0	0	-1
3	0	3	0	0	0	-0
4	12	0	9	0	0	-0
5	11	0	0	0	0	-0
	0	1	0	0	0	

$$g_i: g_{cost} + c_{cost} = 12 + 0 = 12$$

$$= 12$$

$g_i$	1	2	3	4	5	
1	0	0	0	0	0	-0
2	12	0	11	0	0	-1
3	0	3	0	0	0	-0
4	12	0	9	0	0	-0
5	11	0	0	0	0	-0
	0	1	0	0	0	

$$g_i: g_{cost} + c_{cost} = 11 + 0 = 11$$

$$= 11$$

$g_i$	1	2	3	4	5	
1	0	0	0	0	0	-0
2	12	0	11	0	0	-1
3	0	3	0	0	0	-0
4	12	0	9	0	0	-0
5	11	0	0	0	0	-0
	0	1	0	0	0	

$$g_i: g_{cost} + c_{cost} = 12 + 0 = 12$$

$$= 12$$

$g_i$	1	2	3	4	5	
1	0	0	0	0	0	-0
2	12	0	11	0	0	-1
3	0	3	0	0	0	-0
4	12	0	9	0	0	-0
5	11	0	0	0	0	-0
	0	1	0	0	0	

$$g_i: g_{cost} + c_{cost} = 11 + 0 = 11$$

$$= 11$$

$g_i$	1	2</

$x_i$	1	2	3	4	5
1	0	0	0	0	0
2	12	0	12	0	0
3	0	0	0	0	2
4	0	0	0	0	0
5	12	0	0	0	0

$$\hat{C}(2) = C(R) + A(2,3) + \lambda \\ = 28 + 0 + 12 = 50.52$$

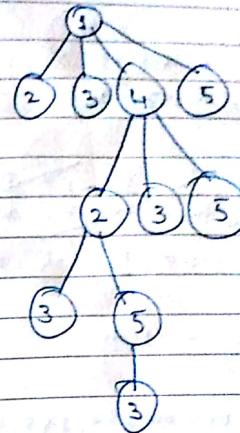
$$\hat{C}(2) = C(R) + A(2,3) + \lambda \\ = 28 + 0 + 0 = 28$$

$x_i$	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	0	0
3	12	0	0	0	2
4	0	0	0	0	0
5	12	0	0	0	0

Minimum cost node = Currently exploring node = 2.5

$$\hat{C}(3) = C(R) + A(3,5) + \lambda = 28 + 0 + 0 = 28.$$

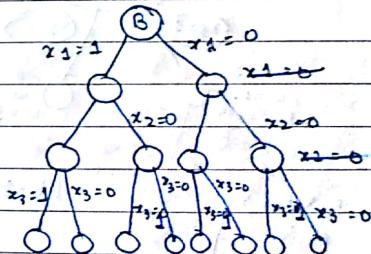
$x_i$	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0



1 → 4 → 2 → 5 → 3

### Knapsack Problem - Backtracking

"In number of items, each item is associated with weight value and profit value."



M = 26

$$W_i = \{13, 12, 8, 7, 9\}$$

$$P_1 = \{23, 24, 25, 26\}$$

$$\frac{P_1}{w_1} = \frac{11}{2}, \quad \frac{P_2}{w_2} = \frac{12}{4}, \quad \frac{P_3}{w_3} = \frac{1}{1}$$

$$\frac{P_1}{W_1} = \frac{23}{11} = 2.09, \quad \frac{P_2}{W_2} = \frac{24}{12} = 2, \quad \frac{P_3}{W_3} = \frac{15}{8} = 1.8, \quad \frac{P_4}{W_4} = \frac{12}{7}$$

$$\frac{P_5}{w_5} = \frac{36}{9} = 3.77$$

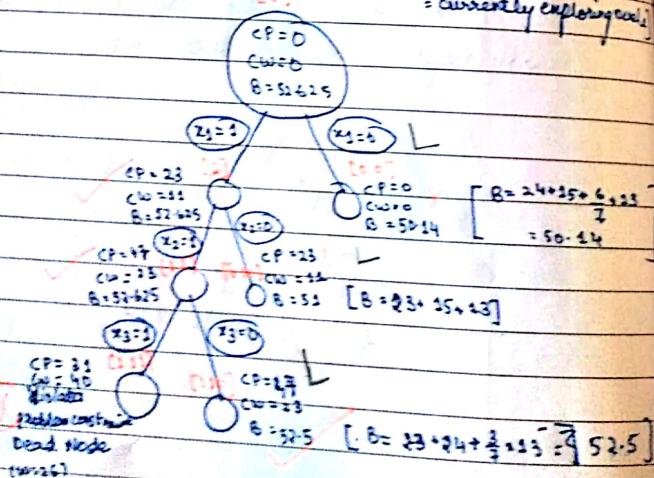
$$P_i = \{2.09, 2.00, 1.82, 1.95, 1.77\}$$

Note 3:

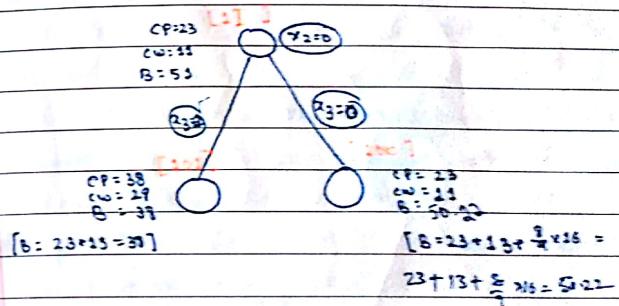
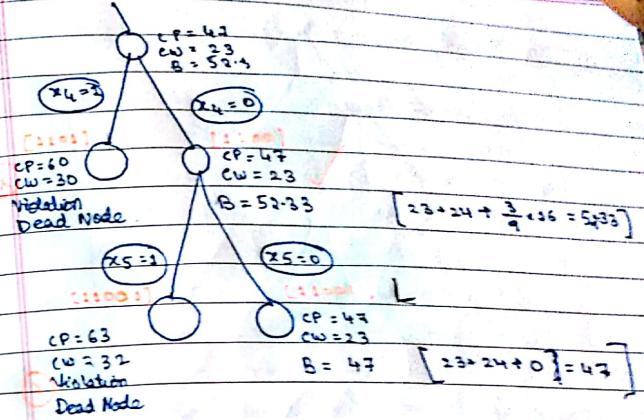
Current Profit = 0, Current Weight = 0, Bounding Function =  $\hat{S}(t_{12})$  (Estimated Budget)

$$\text{By Fractional Knapsack, } 42 + \frac{3}{8} \times 25 = 52.625$$

• our profit node (ie)  
= currently exploring new



1000



## 0/1 Knapsack Problem - Branch and Bound

Branch and Bound Technique solves only minimization problem, but 0/1 knapsack problem is maximization problem. So, we convert maximization problem into minimization problem.

We define two bounding functions : upper bound and lower bound, they are calculated for each node.

For upper bound, fractional part of items are not allowed; i.e. ~~fractional part of items are not allowed~~

For lower bound, fractional part of items are not allowed.

The node which gives least value is currently exploring node.

If for any node, the lower bound value is greater than upper bound value, that node does not generate solution and we are killing it.

To convert into minimization problem, we take upper bound value as negative value.

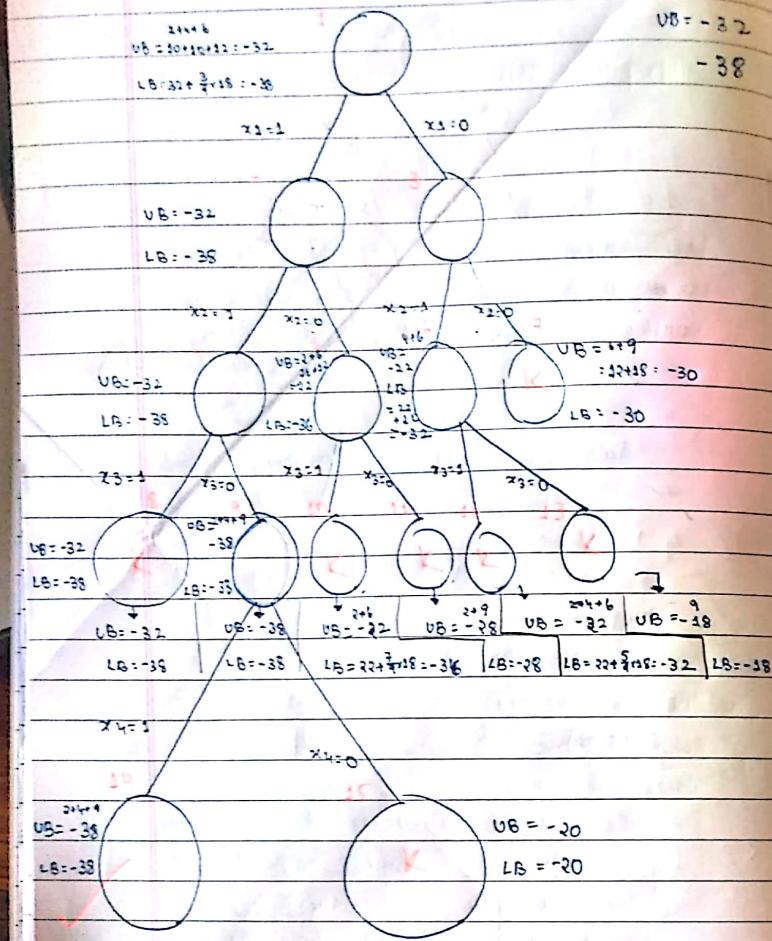
For each node, we calculate the difference between upper bound and lower bound. Which node gives least value is currently exploring node.

We use two ways for this:

i) LC-BB : Find difference of UB and LB, least value is currently exploring node

ii) FIFO : Exploring the nodes in first in first out manner





Path:  $x_1 x_2 x_3 x_4$ : 1101

UNIT-9

N-P  
Polymer: Hard and NP Composite Problems.

→ Algorithms with time complexity polynomial time

- Linear Search -  $O(n)$
- Binary Search -  $O(\log n)$
- Insertion Sort -  $O(n^2)$
- Merge Sort -  $O(n \log n)$
- Matrix Multiplication -  $O(n^3)$
- Strassen's Matrix -  $O(n^{2.37})$

→ Algorithms with time complexity exponential time

- 0/1 Knapsack Problem -  $O(2^n)$
- TVSP -  $O(2^n)$
- Graph Colouring -  $O(2^n)$
- Hamiltonian Cycle -  $O(2^n)$

NP hard and NP complete provides a framework that helps us to solve all problems with exponential time complexity in polynomial time.

If we are not able to write algorithm with polynomial time for algorithms with exponential time, then we must write non-deterministic polynomial algorithm. In non deterministic algorithms we have statements which do not have a clear answer, i.e., yes or no.

If we are not able to write algorithm with polynomial time for algorithms with exponential time then we must find the relation between one problem (that can be solved in polynomial time) and other problems (that cannot be solved in polynomial time).

→ Non deterministic algorithm:

i) For searching algorithm //NP class problem  
search(a, n, key)

j = choice(); // 1

if (a[j] == k)

{

write(a[j])

success(); // 1

}

write(0);

failure(); // 1

}

// O(1)

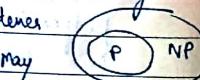
→ Whenever we write non deterministic algorithm, we don't know the exact operation of non-deterministic statements, we leave the blanks, they are filled in the future. (e.g. choice, success, failure)

We have two classes of problems:

Class P: gives polynomial time complexity  
deterministic algorithm

Class NP: gives polynomial time complexity  
non deterministic algorithm

Class P problems are subset of class NP,  
whatever class problems are in class P, today  
they were in class NP yesterday. Whatever  
problems are in class NP today, they may  
be in class NP tomorrow.  $P \subseteq NP$



→ Satisfiability problem

CNF

$$x_i = (x_1, x_2, x_3)$$

$$CNF = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3) \rightarrow \text{Propositional Calculus Formula}$$

→ Assume there are two clauses. We have to verify all the possibilities for  $x_1 x_2 x_3$ , which satisfies the statement

$$x_1 x_2 x_3 \quad 8 = 2^3 = 2^n \text{ possibilities}$$

0 0 0

0 0 1

0 1 0

0 1 1

1 0 0

1 0 1

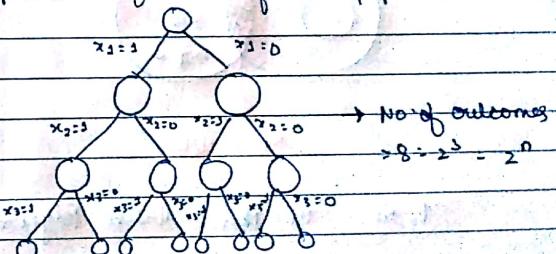
1 1 0

1 1 1

Order of satisfiability problem:  $2^n$

If any person solves satisfiability problem in polynomial time, the all problems with exponential time are solved in polynomial time and related to the satisfiability problem and then solved in exponential time.

State space tree for satisfiability problem.



e.g. 0/1 knapsack problem

$$n = 3$$

$$w_1 \quad w_2 \quad w_3$$

$$x: \{0/1, 0/1, 0/1\}$$

Same state space tree for 0/1 knapsack problem and satisfiability problem. If a researcher solves the satisfiability problem in ~~exponential time~~ polynomial time, then we relate the satisfiability problem to the 0/1 knapsack problems and then to all other algorithms.

It uses Reduction:

Satisfiability of  $L_1$

$L_1 \leq L_2$

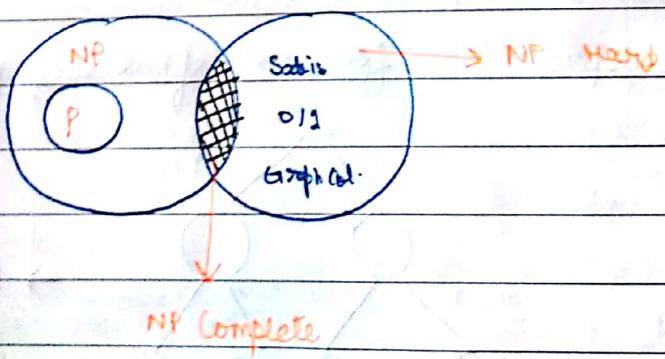
[ $d \rightarrow$  reduction,  $L_1 \rightarrow$  Level 1,  $L_2 \rightarrow$  Level 2]

→ NP Hard and NP Complete Problems

If all problems are NP Hard problems, the problems which use non deterministic

algorithm are called NP Complete problems.

NP complete: NP hard problems which will be solved tomorrow, then include in,  
NP complete



P = NP      NP = Non recursive polynomial algorithm