Consider the following variant of the Vertex Cover problem: we are given a graph G = (V, E). Each edge carries a positive penalty $p_e > 0$ if not covered. Each vertex carries a total cost $c_v > 0$. We want to find a set S of vertices such that the sum of total costs of vertices in S , plus the sum of penalties for edges not covered by S (i.e. no endpoint of e is in S is minimized).

Write an integer formulation for the problem above.

$min(\sum_{v \in V} c_v x_v + \sum_{e \leftarrow (u,v) \in E} p_e x_e)$

$x_u + x_v + x_e \geq 1, \ \forall e \leftarrow (u,v) \in E$

$x_v \in \{0,1\}, \forall v \in V$

$x_e \in \{0,1\}, \forall e \in E$

Consider the following algorithm: Solve the linear programming relaxation of the IP. Choose appropriately a value Δ > 0. Round up to 1 all variables of the LP if their OPT value is ≥ Δ, down to 0. Prove that if we choose appropriately the value of Δ (to what value ?) this algorithm is a 3-approximation algorithm for the problem above.

Hint: This is similar to the arguments for the Vertex Cover problem.

Let $\overline{x}$ an optimal solution to the LP. We round the value to the closest integer as follows:

$x' = 1$ if $\overline{x} \geq \Delta = \frac{1}{3}$ and

$x' = 0$ if $\overline{x} < \Delta = \frac{1}{3}$.

$\sum_{v \in V} c_v x_v + \sum_{e \leftarrow (u,v) \in E} p_e x_e$
$= \sum_{v \in V} c_v \overline{x}_v + \sum_{e \leftarrow (u,v) \in E} p_e \overline{x}_e$
$\leq \sum_{v \in V} 3c_v x'_v + \sum_{e \leftarrow (u,v) \in E} 3p_e x'$
$= 3 \sum_{v \in V} c_v x'_v + 3 \sum_{e \leftarrow (u,v) \in E} p_e x$
$= 3 \cdot OPT(LP)$

The solution set is valid because for every $e \leftarrow (u,v) \in E$ we impose to take u or v, otherwise the penalty x_e must pe added. So, there is at least one variable of $\overline{x}_u, \overline{x}_v, \overline{x}_e$ to be at least $\frac{1}{3}$.

Consider the following linear programming problem. Write its dual.

$min(y_1 + 2y_2 + 3y_3)$ subject to the constraints

$4y_1 - 5y_2 + 6y_3 \geq 7$

$-8y_1 + 9y_2 + 10y_3 \geq 11$

$12y_1 + 13y_3 \geq 4$

$15y_1 + 16y_2 + 17y_3 \geq 18$

$y_i \geq 0,\ for\ i = 1, 2, 3$

The dual problem:

$max(7x_1 + 11x_2 + 4x_3 + 18x_4)$

$4x_1 - 8x_2 + 12x_3 + 15x_4 \leq 1$

$-5x_1 + 9x_2 + 16x_4 \leq 2$

$6x_1 + 10x_2 + 13x_3 + 17x_4 \leq 3$

$x_i \geq 0,\ for\ i = 1, 2, 3, 4$

Apply the two-phase simplex algorithm to the following problem.

$max(4y_2 - y_1)$, subject to

$y_1 + 4y_2 \leq 12$

$y_2 \leq 2$

$y_1 \leq 6$

$y_1, y_2 \geq 0$

We first add the slack variables.

$y_1 + 4y_2 + y_3 = 12$

$y_2 + y_4 = 2$

$y_1 + y_5 = 6$

$and\ y_1, y_2, y_3, y_4, y_5 \geq 0$

We have an initial feasible solution (0, 0, 0, 0, 0).

#	y1	y2	y3	y4	y5	ratio
12	1	4	1	0	0	12/4 = 3
2	0	1	0	1	0	2/1 = 2
6	1	0	0	0	1	
C	-1	4	0	0	0	
Z	0	0	0	0	0	
Z - C	1	-4	0	0	0	

We have -4 as the negative minimum on the second column and 2 as the minimum ratio on the second row. The entering variable is y_2 and the leaving basis y_4.

We apply the following transformation: $r1 \leftarrow r1 - 4r_2$

#	y1	y2	y3	y4	y5
4	1	0	1	-4	0
2	0	1	0	1	0
6	1	0	0	0	1
C	-1	4	0	0	0
Z	0	4	0	4	0
Z - C	1	0	0	4	0

We found the optimal solution as :

$$y_1 = 0, y_2 = 2$$
$$max(4y_2 - y_1) = 8$$

Using either Python/Pulp or Julia/JuMP, submit a program that solves the following problem:

Place 8 queens on an 8x8 board such that the queens don't attack eachother and the sum of the y coordinate values of the queens is minimized.

You may use any solver you want. In case the model takes too much time and does not complete, simply describe the model and the experimental testing conditions.

In order to solve this problem we will use 8x8 binary variables. Since we have N queens on a NxN board, it means that we have a single queen on each row, and a single queen on each column. So, the sum of the binary variables on each row and each column is 1. In the case of diagonals, we can have at most one queen on a diagonal, because there also is the possibility that the diagonal is empty.

In [6]:

```
from pulp import *
import numpy as np

prob = LpProblem("N-Queens", LpMinimize)
table_size = 8

queens = np.array([[
    LpVariable("q_{0}{1}".format(i, j), 0, 1, LpInteger)
     for j in range(table_size)]
        for i in range(table_size)
])
queens
```

Out[6]:

```
array([[q_00, q_01, q_02, q_03, q_04, q_05, q_06, q_07],
       [q_10, q_11, q_12, q_13, q_14, q_15, q_16, q_17],
       [q_20, q_21, q_22, q_23, q_24, q_25, q_26, q_27],
       [q_30, q_31, q_32, q_33, q_34, q_35, q_36, q_37],
       [q_40, q_41, q_42, q_43, q_44, q_45, q_46, q_47],
       [q_50, q_51, q_52, q_53, q_54, q_55, q_56, q_57],
       [q_60, q_61, q_62, q_63, q_64, q_65, q_66, q_67],
       [q_70, q_71, q_72, q_73, q_74, q_75, q_76, q_77]], dtype=obje
ct)
```

In [7]:

```
cost = [
    [j for j in range(table_size)]
        for i in range(table_size)
]
cost
```

Out[7]:

```
[[0, 1, 2, 3, 4, 5, 6, 7],
 [0, 1, 2, 3, 4, 5, 6, 7],
 [0, 1, 2, 3, 4, 5, 6, 7],
 [0, 1, 2, 3, 4, 5, 6, 7],
 [0, 1, 2, 3, 4, 5, 6, 7],
 [0, 1, 2, 3, 4, 5, 6, 7],
 [0, 1, 2, 3, 4, 5, 6, 7],
 [0, 1, 2, 3, 4, 5, 6, 7]]
```

In [8]:

```
# objective function
prob += lpSum([
    cost[i][j] * queens[i][j]
        for i in range(table_size)
            for j in range(table_size)
])

for i in range(table_size):
    # for rows
    prob += lpSum(queens[i]) == 1
    # for columns
    prob += lpSum(queens[:, i]) == 1

diags = [
    queens[::-1,:].diagonal(i)
        for i in range(-queens.shape[0] + 1, queens.shape[1])
]
diags.extend(
    queens.diagonal(i)
        for i in range(queens.shape[1] - 1,-queens.shape[0], -1)
)

# diagonals
for diag in diags:
    prob += lpSum(diag) <= 1

prob.solve()

print("Status:", LpStatus[prob.status])

for index, v in enumerate(prob.variables()):
    print(
        'O|' if v.varValue == 0 else 'X|',
        end = '\n' if (index + 1) % table_size == 0 else ''
    )

print("Min Sum of Y's = ", value(prob.objective))
```

```
Status: Optimal
O|X|O|O|O|O|O|O|
O|O|O|O|O|O|O|X|
O|O|O|O|O|X|O|O|
X|O|O|O|O|O|O|O|
O|O|X|O|O|O|O|O|
O|O|O|O|X|O|O|O|
O|O|O|O|O|O|X|O|
O|O|O|X|O|O|O|O|
Min Sum of Y's =  28.0
```

Present, in the context of a concrete linear programming example (other from ones discussed in class) a cover cut.

What are cuts good for ?

Cuts are stronger constraints that can help solving complex MILP problems by reducing the feasible region, thus making them simpler to solve. The set of feasible solutions is not affected in the integer programming problem.

When we have a constraint in the form of a knapsack constraint we can add cover cuts. The subset of the variables which would violate the inequality if they were all set with 1, would form a minimal cover, but only if we can exclude a single element from the set, the initial inequality must hold.

Example: $2x_1 + x_2 + 5x_3 \geq 6$

$x_1, x_2, x_3 \in \{0, 1\}$

Let C = {1,2} be our minimal cover.

Then $b - \sum_{k \notin C} a_k = 6 - a_3 = 6 - 5 = 1$

Since $a_1 = 2 \geq 1$ and $a_2 = 1 \geq 1$ the corresponding cover cut is $x_1 + x_2 \geq 1$

You want to order catering for a dinner for 140 programmers. You may order any number of small plateaus, that serve 4 people and cost 50 euros per plateau, medium plateaus that serve 9 people and cost 90 euros per plateau and large plateaus, that serve 12 people and cost cost 110 euro. Each plateau is to be placed on a table (small, medium or large). One can order at most 10 plateaus of each type. Give a branch-and-bound approach to compute the minimum number of table needed of each type, while minimizing the total cost of food. You may use a program (Python/Pulp or Julia/Jump) to (optimally) solve the relaxations appearing in the B&B approach

The problem can be formulated accordingly:

$min(50x_0 + 90x_1 + 110x_2)$ subject to

$4x_0 + 9x_1 + 12x_2 \geq 140$

$x_i \leq 10$

$x_i \geq 0$

$x_i \in \mathbb{Z}\ for\ i = 0, 1, 2$

Solving the relaxation implies that the last constraint is not taken into consideration. A B&B approach would be solving first the relaxation and then add new branches with two new constraints on the variable with the biggest fractional part. If all variables are integers, we found the optimum, if not, we compute the current solution. Since it's a minimisation problem, we round the variables up. The current relaxation solution is our lower bound and the current (rounded) solution is our upper bound. We want to improve the current bounds as much as we can. If x'_j is our variable with the largest fractional value, then we solve two more problems, each with a different additional constraint $x_j >= [x'_j]$ and $x_j <= [x'_j]$. We apply this algorithm on each branch until we reach to infeasible solutions, or the upper bounds are not improving.

Next, we will compute the nodes in the tree using PuLP, reaching to the minimum cost of 1330 EUR with 13 tables (1 small, 2 medium, 10 large).

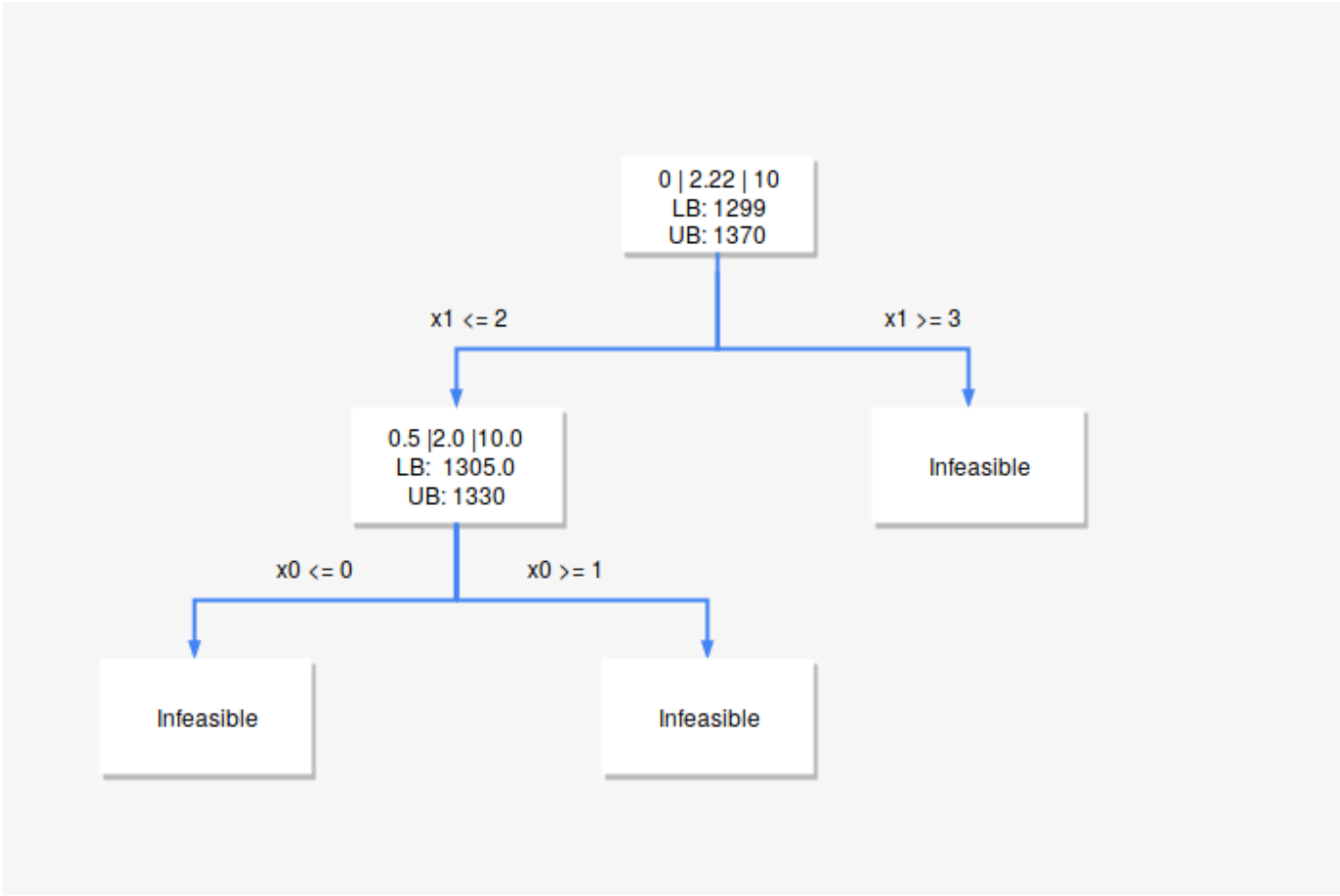
In [95]:

```
from pulp import *
import numpy as np

prob = LpProblem("Catering", LpMinimize)
cost = [50, 90, 110]
people = [4, 9, 12]

x = [LpVariable("x_{0}".format(i), 0, None) for i in range(3)]
prob += lpSum([cost[i] * x[i] for i in range(3)])
prob += lpSum([people[i] * x[i] for i in range(3)]) >= 140
for i in range(3):
        prob += x[i] <= 10

def solve_lp_relaxation(additional_constraints = []):
    global prob, x
    for constraint in additional_constraints:
        prob += constraint
    prob.solve()
    print("Status:", LpStatus[prob.status])
    for index, v in enumerate(prob.variables()):
        print(v.varValue)
    print("Min cost = ", value(prob.objective))
```

In [96]:

```
solve_lp_relaxation()
solve_lp_relaxation([x[1] <= 2])
solve_lp_relaxation([x[1] >= 3])
solve_lp_relaxation([x[1] <= 2, x[0] <= 0])
solve_lp_relaxation([x[1] <= 2, x[0] >= 1])
```

```
Status: Optimal
0.0
2.2222222
10.0
Min cost =  1299.999998
Status: Optimal
0.5
2.0
10.0
Min cost =  1305.0
Status: Infeasible
0.0
3.0
9.4166667
Min cost =  1305.833337
Status: Infeasible
0.0
3.0
9.4166667
Min cost =  1305.833337
Status: Infeasible
0.0
3.0
9.4166667
Min cost =  1305.833337
```


0 | 2.22 | 10
LB: 1299
UB: 1370
x1 <= 2
x1 >= 3
0.5 |2.0 |10.0
LB: 1305.0
UB: 1330
Infeasible
x0 <= 0
x0 >= 1
Infeasible
Infeasible