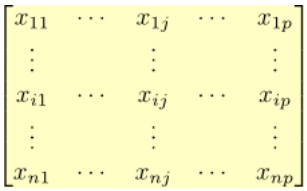
ソースコード解説

美術館のマス目を下の図のように添字を置いて考える（n→I, p→J）



- `Museum()`クラス

  美術館全体のクラス，美術館の大きさ，壁の数の情報と各エリアの情報を二次元配列として持つ

  `museum.I`：美術館の縦の広さ（行の数）

  `museum.J`：美術館の横の広さ（列の数）

  `museum.room`：美術館全体（行列）を二重配列で表現，$I \times J$の大きさのマス目

- `Area(Museum)`クラス

  `Museum()`の子クラスでエリアの種類と要素を持つ

  - `area.label`：エリア（マス）の種類

    壁："wall"

    廊下："hallway"

  - `area.prop`：エリアのプロパティ

    壁：`None`，`0` ~ `4`（隣接する照明の制約）

    廊下：`0` or `1`

> 1マスが持つ情報が多かったのでリストや辞書じゃなくてクラスを定義した

- `addRequirements(i, j, num, museum)`

  問題を設定する関数，`num`は制約の数字（`None` or `0` ~ `4`）

- `getAroundLights(i, j, museum)`関数

  x[i, j]マスに隣接した4箇所の照明の数を数える
  上下左右のマスについて，labelが"hallway"である時propをリストに追加し，その合計値を返す
  ※ この時propは0-1なのでリストの合計が周辺の照明の数の同じになる

- `getVLightsIlluminated(i, j ,museum)`関数

  行を見て壁までにある照明の数を数える

- `getHLightsIlluminated(i, j, museum)`関数

  列を見て壁までにある照明の数を数える

- 制約

```
# 制約
for i in range(0,museum.I):
    for j in range(0,museum.J):

        # 壁の制約
```

```
        if museum.room[i][j].label == "wall":
            if museum.room[i][j].prop == None:
                pass
            else:
                # 壁に接するエリアには数字の数だけライトをおく
                model.addConstr(museum.room[i][j].prop == getAroundLights(i,j,museum), "wall(%s,%s)_constraint"%(i+1,j+1))

        # ライトの制約
        elif museum.room[i][j].label == "hallway":
            # ライトはライトで照らさない(行)
            model.addConstr(quicksum(getVLightsIlluminated(i,j,museum)) <= 1, "light(%s,%s)_constraint_1"%(i+1,j+1))
            # ライトはライトで照らさない(列)
            model.addConstr(quicksum(getHLightsIlluminated(i,j,museum)) <= 1, "light(%s,%s)_constraint_2"%(i+1,j+1))
            # 廊下の制約（どの廊下も照らされる）
            model.addConstr(
                quicksum(getVLightsIlluminated(i,j,museum)) + quicksum(getHLightsIlluminated(i,j,museum)) - museum.room[i][j].prop >= 1
                "hallway(%s,%s)_constraint"%(i+1,j+1)
            )
```


制約は1マスごとに付与した
照明の数を数える時，行と列で制約を分けたのは，
ある1マスからみてその直線上に照明が重なっているかを調べるため（縦方向と横方向の両方から照らされても良いため）