

# 数理最適化によるパズルの解法

## Agenda

- 自己紹介
- オペレーションズ・リサーチ(OR)とは
- 数理最適化とは
- 数独
- カックロ
- ののぐらむ
- まとめ

## 自己紹介

- 名前: 斎藤努
- 会社: 構造計画研究所
- 仕事: オペレーションズ・リサーチ(主に最適化)を使った、コンサルや開発をしています

## オペレーションズ・リサーチ(OR)とは

数学を使って様々な問題を解決する学問です。いろいろなテーマがあります。

- 数理最適化
  - 線形計画問題
  - 混合整数計画問題 (←今日の話)
- シミュレーション
- 待ち行列
- PERT
- AHP(階層的意思決定モデル)
- スケジューリング
- ゲーム理論などなど

## ORの心得

- プロフェッショナルとしての自覚を持ち、真摯に行動しましょう
- 目的をはっきりさせます
  - 進む方向を定めます
- モデルを作り解きます
  - モデルはシンプルにしましょう
  - 現実の8割を表せれば十分なこともあります
  - モデルを複雑にするより全体を把握できることが大切です
- 答えが合っていないかったら戻りましょう
  - 前提を見直しましょう
  - モデルと結果の関係から得られるものが重要です
  - 答えがあっているなら説得しなければいけません
- 適用して目的を達成します
  - 効果が出ることで意味があります

# 数理最適化とは

- 数理最適化では、問題を 数理モデル で表して、それを解きます
  - 数理モデルは、非常にシンプルなルールで様々な問題を記述できます
  - 解ぐソフトウェアを ソルバ とよびます
- 特定の問題は専用のソルバを使うこともありますが、汎用のソルバを使うことが多いです
  - 今回使うソルバは、汎用ソルバでCBCという無料のソルバです
  - モデラーがpulpになります

## pulpとは

- Pythonによる数理最適化モデリングツールです
- Gurobi, CBC, GLPKなど複数のソルバが使えます
- インストールは、setuptoolsがあれば、「easy\_install pulp」です
- 「from pulp import \*」で利用可能
- BSD 3-Clause License

## pulpの数理モデル作成

- 最初に、数理モデル(**LpProblem**)を作成します
- 次に、必要な分だけ、変数(**LpVariable**)を追加します
- 次に、目的関数の式(**LpAffineExpression**)を追加します
- 最後に、制約(**LpConstraint**)を追加します

## pulpの数理モデル(**LpProblem**)

- コンストラクタ LpProblem(self, name='NoName', sense=LpMinimize)
  - name:数理モデル名
  - sense:最小化(LpMinimize)または最大化(LpMaximize)
- solve(solver)
  - solverを用いて問題を解く。solver省略時は、cbcを用いる。GLPKの場合、solve(solvers.GLPK())

```
In [1]: from pulp import *
LpProblem[1]
```

## pulpの変数(**LpVariable**)

- コンストラクタ LpVariable(self, name, lowBound=None, upBound=None, cat='Continuous', e=None)
  - name:変数名
  - lowbound:下限
  - upbound:上限
  - cat:種類
    - LpContinuous:連続変数
    - LpInteger:整数変数
    - LpBinary:バイナリ変数(0または1)

In [2]: LpVariable?

## pulpの式(LpAffineExpression)

- 変数を使って「 $x+2*y$ 」のように作成可能
- 式に後から項を追加するのも可能

## pulpの制約(LpConstraint)

- 変数を使って「 $x+y \leq 1$ 」のように作成可能
- 制約の左辺に後から項を追加するのも可能

## pulpのよく使う関数

- value():変数の値を取り出します
- lpSum():項の和を求めます
- lpDot():2つのリストの内積を求めます
- 参考:<http://mathopt.sakura.ne.jp/pythonpulp.html> (<http://mathopt.sakura.ne.jp/pythonpulp.html>)

「数独」「カックロ」「ナンバーリンク」「スリザーリンク」「推理パズル」「ましゅ」はニコリの登録商標です

出典 ニコリ<http://www.nikoli.co.jp/ja/> (<http://www.nikoli.co.jp/ja/>)

## 利用できるソルバーの一覧表示

```
In [36]: from pulp import *
def AvailableSolver(cls):
    for c in cls.__subclasses__():
        try:
            AvailableSolver(c)
            if c().available(): print(c)
        except NotImplementedError: pass
AvailableSolver(LpSolver)
```

```
<class 'pulp.solvers.GLPK_CMD'>
<class 'pulp.solvers.PULP_CBC_CMD'>
```

## 変数作成用の関数を定義(手間を省くため)

```
In [1]: from __future__ import print_function
from pulp import *
# 準備
# 実数変数作成関数
def addvar(s='v', cat=LpContinuous, n=[0]):
    n[0] += 1
    return LpVariable(s + str(n[0]), lowBound=0, cat=cat)

# バイナリ変数作成関数
def addbinvar(s='v'): return addvar(s, cat=LpBinary)
```

## 数独の解き方

データ (data/sudoku.txt)

### 問題

- マスに 1~9 の数字を入れます
- 縦、横または  $3 \times 3$  に同じ数字は入りません

		6						1
	7			6				5
8			1		3	2		
		5		4		8		
	4		7		2		9	
		8		1		7		
		1	2		5			3
	6			7			8	
2						4		

### 定式化

$$\begin{aligned}
 \text{変数} \quad & v_{ijk} \in \{0, 1\} \quad \forall i, j, k \quad \text{マス } i, j \text{ が数字 } k+1 \text{ か} \\
 \text{subject to} \quad & \sum_k v_{ijk} = 1 \quad \forall i, j \quad \text{数字は1つ} \\
 & \sum_k v_{ikj} = 1 \quad \forall i, j \quad \text{縦に同じ数字はない} \\
 & \sum_k v_{kij} = 1 \quad \forall i, j \quad \text{横に同じ数字はない} \\
 & 3 \times 3 \text{ のマスについても同様} \\
 & \text{数字指定}
 \end{aligned}
 \tag{1} \tag{2} \tag{3} \tag{4} \tag{5} \tag{6}$$

```
In [2]: r3, r9 = range(3), range(9)
m = LpProblem()
v = {(i, j, k):addbinvar() for i in r9 for j in r9 for k in r9} # (1)
with open('data/sudoku.txt') as fp:
    for i in r9:
        s = fp.readline()
        for j in r9:
            if s[j].isdigit():
                m += v[i, j, int(s[j]) - 1] == 1 # (6)
for i in r9:
    i0, j0 = i // 3, i % 3
    for j in r9:
        m += lpSum(v[i, j, k] for k in r9) == 1 # (2)
        m += lpSum(v[i, k, j] for k in r9) == 1 # (3)
        m += lpSum(v[k, i, j] for k in r9) == 1 # (4)
        m += lpSum(v[i0 * 3 + i1, j0 * 3 + j1, j] for i1 in r3 for j1 in r3) == 1
# (5)
%time m.solve()
for i in r9:
    for j in r9: print(sum(k * int(value(v[i, j, k])) + 0.5) for k in r9) + 1, end
    print()
```

Wall time: 112 ms

5	3	6	8	2	7	9	4	1
1	7	2	9	6	4	3	5	8
8	9	4	1	5	3	2	6	7
7	1	5	3	4	9	8	2	6
6	4	3	7	8	2	1	9	5
9	2	8	5	1	6	7	3	4
4	8	1	2	9	5	6	7	3
3	6	9	4	7	1	5	8	2
2	5	7	6	3	8	4	1	9

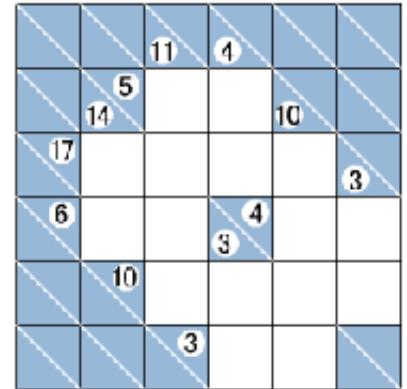
まとめ

# カックロの解き方

データ ([data/kakkuro.txt](#))

## 問題

- マスに 1~9 の数字を入れます
- 縦または横に連続する空マスの中に同じ数字は入りません
- 連続するマスの合計が示されます



## 定式化

$$\text{変数} \quad v_{ijk} \in \{0, 1\} \quad \forall i, j, k \quad \text{マス } i, j \text{ が数字 } k+1 \text{ か} \quad (1)$$

$$r_{ij} \in Z \quad \forall i, j \quad \text{マス } i, j \text{ の数字} \quad (2)$$

$$\text{subject to} \quad \sum_k v_{ijk} = 1 \quad \forall i, j \quad \text{数字は1つ} \quad (3)$$

$$\sum_k k \times v_{ijk} + 1 = r_{ij} \quad \forall i, j \quad r \text{を } v \text{ で表現} \quad (4)$$

$$\sum_{ij} r_{ij} = \text{合計} \quad \forall i, j \quad \text{マスの合計が同じ} \quad (5)$$

マスの並びで同じ数字は禁止 (6)

```
In [3]: with open('data/kakkuro.txt') as fp:
    nw, nh = [int(s) for s in fp.readline().split(',')]
    rw, rh, r2, r9 = range(nw), range(nh), range(2), range(9)
    ch = [fp.readline() for i in rh]
    hh = fp.readlines()
m = LpProblem()
v = {(i, j, k): addbinvar() for i in rw for j in rh for k in r9} # (1)
r = {(i, j): addvar('r') for i in rw for j in rh} # (2)
cnt = -1
for j in rh:
    for i in rw:
        if ch[i][j] != '*':
            m += lpSum(v[i, j, k] for k in r9) == 1 # (3)
            m += lpSum(k * v[i, j, k] for k in r9) + 1 == r[i, j] # (4)
            continue
        cnt += 1
        nn = [int(t) if t.isdigit() else -1 for t in hh[cnt].rstrip('\n').split(
            '$')] # (5)
        for drc in r2: # 縦と横
            if nn[drc] < 0: continue
            x, y = i, j
            l = []
            while True:
                x, y = x + drc, y + 1 - drc
                if x == nw or y == nh or ch[x][y] == '*': break
                l.append((x, y))
            for h in r9:
                m += lpSum(v[x, y, h] for x, y in l) <= 1 # 並びで同じ数は1つ (6)
                m += lpSum(r[x, y] for x, y in l) == nn[drc] # 合計が等しい (5)
%time m.solve()
for j in rh:
    for i in rw:
        print('*' if ch[j][i] == '*' else '%d' % int(value(r[i, j]) + 0.5), end=' ')
    print()
```

Wall time: 53 ms

```
*****
* * * 9 8 *
* * 3 1 5 2
* 1 2 * 9 7
* 3 4 1 7 *
* * 1 2 * *
```

まとめ

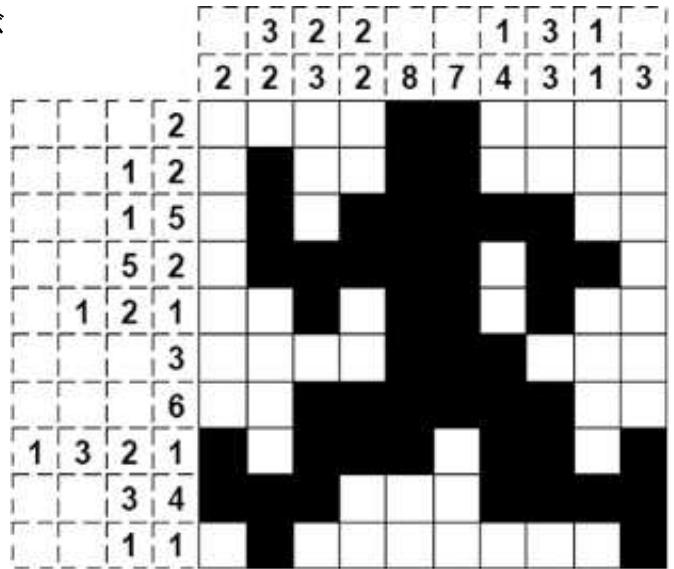
# ののぐらむの解き方

データ (data/nonogram.txt)

## 問題

- 各横行の左、各縦列の上にある数字は、その行(列)の中で連続して黒く塗る白マスの数を表します
- 1つの行(列)に対して数字が複数ある場合は、数字の並び順どおりにその数字の数だけ連続して黒く塗ります
- 1つの行(列)に対して数字が複数ある場合は、その数字が表す黒マスの連続の間に1マス以上の白マス(塗らないマス)が入ります

## 定式化



変数  $v_{ij} \in \{0, 1\} \forall i, j$  マス*i, j*が黒かどうか(1)

$r_k \in \{0, 1\} \forall k$ , 縦または横 縦または横ごとに*k*番目の候補を選ぶかどうか(2)

subject to  $\sum_k r_k = 1 \forall$ 縦または横 縦または横ごとに候補の中から1つ(3)

候補を選んだらマスの色は候補の通り(4)

```
In [4]: def baselist(i, j, k): return [0] * i + [1] * j + [0] * k
def makelist(n, l):
    p = l[-1]
    if len(l) == 1:
        if n < p: return None
        return [baselist(i, p, n - p - i) for i in range(n - p + 1)]
    ll = l[:-1]
    s = sum(ll) + len(ll) - 1
    return [j + baselist(1, p, n - p - s - i - 1) +
            for i in range(n - p - s) for j in makelist(i + s, ll)]
def do(m, v, fp, isw, n1, n2):
    for i in range(n1):
        ss = [int(s) for s in fp.readline().split(',')]
        l = makelist(n2, ss)
        r = [addbinvar('r') for c in l] # (2)
        m += lpSum(r) == 1 # (3)
        for j, c in enumerate(l):
            for k, b in enumerate(c):
                vv = v[i][k] if isw else v[k][i]
                m += (1 - 2 * b) * vv <= 1 - b - r[j] # (4)
m = LpProblem()
with open('data/nonogram.txt') as fp:
    nw, nh = [int(s) for s in fp.readline().split(',')]
    v = [[addvar() for i in range(nw)] for j in range(nh)] # (1)
    do(m, v, fp, True, nw, nh)
    do(m, v, fp, False, nh, nw)
%time m.solve()
for j in range(nh):
    for i in range(nw):
        print('.' * [int(value(v[i][j]) + 0.5)], end=' ')
    print()
```

Wall time: 266 ms

```
. . . . * * . . .
. * . . * * . . .
. * . * * * * * .
. * * * * * . * *
. . * . * * . * .
. . . . * * * . .
. . * * * * * . .
* . * * * . * * . *
* * * . . . * * * *
. * . . . . . . *
```

まとめ

# 美術館の解き方

データ ([data/museum.txt](#))

## 問題

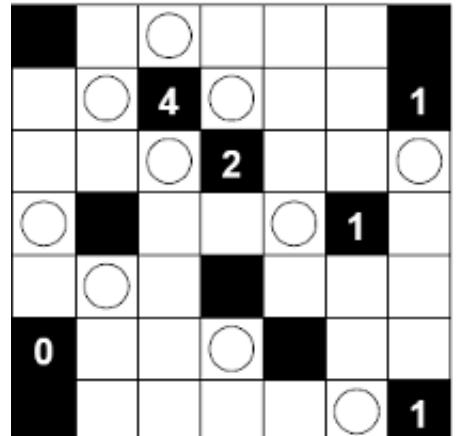
- 黒マスに入っている数字は、それと隣接する縦横両隣の最大4つの白マスに入る○の数を表します
- 照明は、そのマスから上下左右に黒マスか外枠にぶつかるまでの範囲を照らします
- 斜めには照らすことはできません
- どの照明にも照らされていない白マスがあつてはいけません
- 照明のあるマスは他の照明で照らされてはいけません

## 変数

- v:各マスに照明をおくかどうか (1)

## 制約

- 白マスの並びの中で照明は1つ以下 (2)
- 各マスは1つ以上の照明で照らされること (3)
- 数字の周りに同じ数の照明 (4)
- 数字に照明はおけない (5)



```

In [5]: with open('data/museum.txt') as fp:
    nw, nh = [int(s) for s in fp.readline().split(',')]
    rw, rh, r2 = range(nw), range(nh), range(2)
    ch = [fp.readline() for i in range(nh)]
m = LpProblem()
v = [[addbinvar('v') for j in rh] for i in rw] # (1)
e = [[LpAffineExpression() for j in rh] for i in rw]
lst = []
def addlist(s, i, j, x, y, c): # 白マスの並びを返す
    for k in range(len(s)):
        if s[k] == '.':
            c.append((i + x * k, j + y * k))
        else:
            if len(c) > 1: lst.append(c)
            c = []
for j in rh: addlist(ch[j], 0, j, 1, 0, [])
for i in rw: addlist([ch[j][i] for j in rh] + ['n'], i, 0, 0, 1, [])
for l in lst:
    m += lpSum(v[x][y] for x, y in l) <= 1 # (2)
    for x, y in l: e[x][y] += lpSum(v[p][q] for p, q in l)
def dirs(i, j):
    return [v[i + x - y][j + x + y - 1] for x in r2 for y in r2] #
        if 0 <= i + x - y < nw and 0 <= j + x + y - 1 < nh]
for j in rh:
    for i in rw:
        if ch[j][i] == '.':
            m += e[i][j] >= 1 # (3)
        else:
            m += v[i][j] == 0 # (5)
            if ch[j][i].isdigit():
                m += lpSum(dirs(i, j)) == int(ch[j][i]) # (4)
%time m.solve()
for j in rh:
    for i in rw:
        print(ch[j][i] if ch[j][i] != '.' else '+' if value(v[i][j]) > 0.5 else ',',
              end=' ')
    print()

```

Wall time: 38 ms

```

# +
# +
+ 4 + 1
+ 2 +
+ # + 1
+ #
0 + #
# + 1

```

まとめ

# ナンバーリンクの解き方

データ (data/numberlink.txt)

## 問題

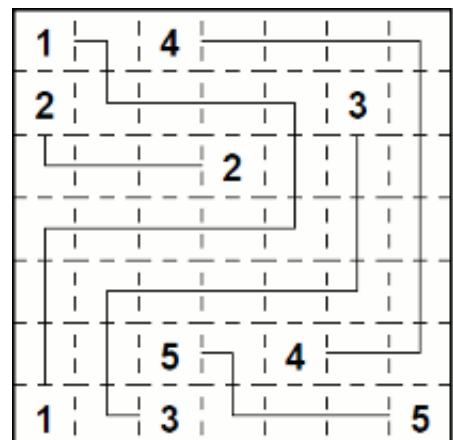
- 各マスには2つ以上の線が入ってはいけません
- 異なる線同士が交わってはいけません

## 変数

- vr:どのタイプのラインに含まれるか (1)
- vh:水平に出るかどうか (2)
- vv:垂直に出るかどうか (3)

## 制約

- 始点と終点が指定のラインに含まれること (4)
- 始点と終点は、1本だけ出ること (5)
- 始点終点以外の各マスに接続されるのは、0本か2本 (6) --- バイナリ変数なしで記述可能！
- 接続したら、両端のタイプは同じになること (7)



```
In [6]: with open('data/numberlink.txt') as fp:
    nw, nh = [int(s) for s in fp.readline().split(',')]
    rw, rh, r2 = range(nw), range(nh), range(2)
    ch = [fp.readline() for i in range(nh)]
    m = LpProblem()
    vr = [[addvar('r') for j in rh] for i in rw] # (1)
    vh = [[addbinvar('h') for j in rh] for i in range(nw - 1)] # (2)
    vv = [[addbinvar('h') for j in range(nh - 1)] for i in rw] # (3)
    def dirs(i, j):
        return [vh[i - k][j] for k in r2 if 0 <= i - k < nw - 1] + [
            vv[i][j - k] for k in r2 if 0 <= j - k < nh - 1]
    for i in rw:
        for j in rh:
            s = dirs(i, j)
            if ch[j][i].isdigit():
                m += vr[i][j] == int(ch[j][i]) # (4)
                m += lpSum(s) == 1 # (5)
            else:
                #m += lpSum(s) == 2 * m.addbinvar() # inefficient (6)
                m += lpSum(s) <= 2 # (6)
                for k in range(len(s)):
                    m += lpSum(s) >= 2 * s[k] # (6)
            if i < nw - 1:
                m += vr[i][j] <= vr[i + 1][j] + 10 * (1 - vh[i][j]) # (7)
                m += vr[i + 1][j] <= vr[i][j] + 10 * (1 - vh[i][j]) # (7)
            if j < nh - 1:
                m += vr[i][j] <= vr[i][j + 1] + 10 * (1 - vv[i][j]) # (7)
                m += vr[i][j + 1] <= vr[i][j] + 10 * (1 - vv[i][j]) # (7)
    %time m.solve()
    for j in rh:
        for i in rw:
            sys.stdout.write('%d%c' % (value(vr[i][j]), '-' if
                                         i < nw - 1 and value(vh[i][j]) >= 0.5 else ' '))
    if j == nh - 1: break
    sys.stdout.write('\n')
    for i in rw:
        sys.stdout.write('%c ' % ('|' if value(vv[i][j]) >= 0.5 else ' '))
    sys.stdout.write('\n')
```

Wall time: 74 ms

1-1-1 2-2-2-2

|

1 4 3-3-3-3-3

| |

1 4 5-5-5 6-6

| | | |

1 4-4-4 5 7 6

| | | |

1 5-5-5-5 7 6

| | | |

1 7-7-7-7-7 6

| |

7-7 6-6-6-6-6

# 覆面算の解き方

データ ([data/fukumen.txt](#))

## 問題

- 各記号は0から9までの数字の1に対応します
  - 同じ記号には同じ数字が対応します
  - 最上位の桁は0になりません

## 変数

- v:各位置でどの数字を使うか (1)
- r:各位置の1桁の数字 (2)
- q:nw桁の数字 (3)

## 制約

- $v_{ij}$  内で数字は1つのみ (4)
- rをvで表現 (5)
- 空白は0、先頭は0でない (6)
- 同じ文字は同じ数字、違う文字は違う数字 (7)
- qをrで表現 (8)
- qの最後以外の合計は、qの最後と等しい (9)

$$\begin{array}{r} \text{S E N D} \\ \text{M O R E} \\ \hline \text{M O N E Y} \end{array} \quad \begin{array}{r} 9 5 6 7 \\ 1 0 8 5 \\ \hline 1 0 6 5 2 \end{array}$$

```
In [7]: with open('data/fukumen.txt') as fp:
    lines = fp.readlines()
    lines = lines[:-2] + [lines[-1]]
    nw, nh = len(lines[0]) - 1, len(lines)
    rw, rh, r10 = range(nw), range(nh), range(10)
    m = LpProblem()
    v = [[[addbinvar('v') for k in r10] for j in rh] for i in rw] # (1)
    r = [[addvar('r') for j in rh] for i in rw] # (2)
    q = [addvar('q') for j in rh] # (3)
    dic = {}
    for j in rh:
        e = LpAffineExpression()
        fst = True
        for i in rw:
            c = lines[j][i]
            e = e * 10 + r[i][j]
            m += lpSum(v[i][j]) == 1 # (4)
            m += lpDot(r10, v[i][j]) == r[i][j] # (5)
            if c == ' ':
                m += v[i][j][0] == 1 # (6)
            else:
                if c in dic:
                    m += lpDot(r10, v[i][j]) == lpDot(r10, dic[c]) # (7)
                else:
                    dic[c] = v[i][j]
            if fst:
                fst = False
            m += v[i][j][0] == 0 # (6)
        m += e == q[j] # (8)
    for i, t in dic.items():
        for j, u in dic.items():
            if i < j:
                for k in r10:
                    m += t[k] + u[k] <= 1 # (7)
    m += lpSum(q[:-1]) == q[-1] # (9)
%time m.solve()
for j in rh:
    if j == nh - 1: print('-' * (nw * 2 - 1))
    for i in rw:
        print('%d' % value(r[i][j]), end=' ')
    print()
```

Wall time: 398 ms

0 9 5 6 7  
0 1 0 8 5

---

1 0 6 5 2

# 不等式の解き方

データ (data/inequality.txt)

## 問題

- 各白マスには1からnまでの数字が1つだけ入れります
  - 各横行および各縦列には同じ数字が入りません
  - 連続する2つのマス目の間に不等号がある場合、それらのマス目に入る数字は不等号の示す大小関係を満たさなければいけません
- 
- The grid consists of 25 small squares arranged in a 5x5 pattern. Some squares contain numbers (1, 2, 3, 4, 5) or symbols (<, >, v, ^). The symbols indicate the relative values of the numbers in adjacent squares. For example, a '<' symbol between two squares means the number in the first square is less than the number in the second square.
- |   |   |   |   |   |
|---|---|---|---|---|
| 5 | 1 | 2 | 3 | 4 |
| 4 | 3 | < | 5 | 1 |
| 3 | < | 4 | 1 | 2 |
| v | 2 | 5 | 3 | 4 |
| 2 | < | 4 | < | 5 |
- |   |   |   |   |   |
|---|---|---|---|---|
| 1 | < | 2 | < | 4 |
| 4 | < | 5 | > | 3 |

## 変数

- v: 各位置がどの数字か (1)
- r: 各位置の数字 (2)

## 制約

- $v_{ij}$  は1つの数字のみ (3)
- rをvで表現 (4)
- 縦または横に同じ数字が入りません (5)
- 数字が指定されていれば、その数字になること (6)
- 不等号があれば、その関係を満たすこと (7)

```
In [8]: with open('data/inequality.txt') as fp:
    n = int(fp.readline())
    rn = range(n)
    lines = fp.readlines()
m = LpProblem()
v = [[[addbinvar('v') for k in rn] for j in rn] for i in rn] # (1)
r = [[addvar('r') for j in rn] for i in rn] # (2)
for j in rn:
    for i in rn:
        m += lpSum(v[i][j]) == 1 # (3)
        m += lpDot(rn, v[i][j]) + 1 == r[i][j] # (4)
        m += lpSum(v[i][k][j] for k in rn) == 1 # (5)
        m += lpSum(v[k][i][j] for k in rn) == 1 # (5)
c = lines[j * 2][i * 2]
if c.isdigit():
    m += v[i][j][int(c) - 1] == 1 # (6)
for j in range(n - 1):
    for i in rn:
        c = lines[j * 2 + 1][i * 2]
        if c == 'A': m += r[i][j] <= r[i][j + 1] - 1 # (7)
        elif c == 'V': m += r[i][j] >= r[i][j + 1] + 1 # (7)
for j in rn:
    for i in range(n - 1):
        c = lines[j * 2][i * 2 + 1]
        if c == '<': m += r[i][j] <= r[i + 1][j] - 1 # (7)
        elif c == '>': m += r[i][j] >= r[i + 1][j] + 1 # (7)
%time m.solve()
for j in rn:
    for i in rn:
        print('%d' % sum((k + 1) * value(v[i][j][k]) for k in rn), end=' ')
    print()
```

Wall time: 53 ms

5 1 2 3 4  
4 3 5 1 2  
3 4 1 2 5  
2 5 3 4 1  
1 2 4 5 3

# ビルディングパズルの解き方

データ ([data/building.txt](#))

## 問題

- 各数字はそのマスに立てられるビルの高さを表します
- 各横行に同じ数字は入りません
- 各縦列に同じ数字は入りません
- 盤面の外側の数字はその数字の書かれている場所から盤面を眺めたときに同じ横行(または縦列)に見えるビルの数を表します

			3	3				3	3		
2							5	2	2	2	5
							1	3	4	4	4
							3	3	1	3	1
							2	3	2	1	5
							2	3	1	2	5
							3	3	3	1	3
							5	5	5	1	1
							1	1	1	5	3
							3	3	3	3	3
							2	2	2	2	2
							5	5	5	1	1

## 変数

- v: 各位置がどの高さか (1)
- r: 各位置の高さ (2)
- u: 各方向別の数字のある各列ごとに (3)

## 制約

- $v_{ij}$  は、1つの高さのみ (4)
- rをvで表現 (5)
- 縦または横に同じ高さがないこと (6)
- 上下左右から見たときにuの合計が「数字-1」(すなわち、高くなるときにu==1とします) (7)

```
In [9]: with open('data/building.txt') as fp:
    n = int(fp.readline())
    rn = range(n)
    lines = fp.readlines()
m = LpProblem()
v = [[[addbinvar('v') for k in rn] for j in rn] for i in rn] # (1)
r = [[addvar('r') for j in rn] for i in rn] # (2)
def add(m, c, p, q, x, y):
    if not c.isdigit(): return
    k = int(c)
    u = [addvar('u') for i in range(n - 1)] # (3)
    m += IpSum(u) == k - 1 # (7)
    vmx = r[p][q]
    for i in rn:
        vnx = r[p + x * i + x][q + y * i + y]
        m += vmx + n * u[i] >= vnx + 1 # (7)
        m += vmx + 1 <= vnx + n - n * u[i] # (7)
        if i == n - 2: break
        vtm = addvar()
        m += vmx <= vtm # (7)
        m += vnx <= vtm # (7)
        vmx = vtm
    for j in rn:
        for i in rn:
            m += IpSum(v[i][j]) == 1 # (4)
            m += IpDot(rn, v[i][j]) + 1 == r[i][j] # (5)
            m += IpSum(v[i][k][j] for k in rn) == 1 # (6)
            m += IpSum(v[k][i][j] for k in rn) == 1 # (6)
    add(m, lines[0][j + 1], j, 0, 0, 1)
    add(m, lines[n + 1][j + 1], j, n - 1, 0, -1)
    add(m, lines[j + 1][0], 0, j, 1, 0)
    add(m, lines[j + 1][n + 1], n - 1, j, -1, 0)
%time m.solve()
for j in rn:
    for i in rn:
        for k in rn:
            if value(v[i][j][k]) > 0.5: print(k + 1, end=' ')
print()
```

Wall time: 1.27 s

2 5 1 3 4  
 5 4 3 2 1  
 4 3 2 1 5  
 1 2 5 4 3  
 3 1 4 5 2

# ウォールロジックの解き方

データ (data/wall.txt)

## 問題

- ・ 数字が記入されているマスからその数字の数だけ縦と横に線を引きます
- ・ 1つのマスには1本の線しか引くことができません
- ・ 数字が記入されているマスには線を引くことができません

4			1		
	4			2	
		2			2
1			1		
	1			1	
		3		2	

4	→	1	→	↑
	4	→	2	
↓		2	→	2
1	↓	1	↓	1
	↓	1	→	1
←	3	→	1	2

## 変数

- ・ v: 各位置のどの方向か (1)
- ・ r: 各位置の方向別長さ (2)

## 制約

- ・ 数字があれば、方向別長さの和に等しく、かつその位置に矢印がないこと (3)
- ・ 数字がなければ矢印は1方向のみ (4)
- ・ 数字がなければ矢印の方向に長さを1足すこと (5)

```
In [10]: with open('data/wall.txt') as fp:
    nw, nh = [int(s) for s in fp.readline().split(',')]
    mx = max(nw, nh)
    rw, rh, r4 = range(nw), range(nh), range(4)
    ch = [fp.readline() for i in range(nh)]
m = LpProblem()
v = [[[addbinvar('v') for k in r4] for j in rh] for i in rw] # (1)
r = [[[addvar('r') for k in r4] for j in rh] for i in rw] # (2)
drc = [(-1, 0, 0), (0, -1, 1), (0, 1, 2), (1, 0, 3)]
def sumdir(i, j):
    return lpSum(r[i + x][j + y][k] for x, y, k in drc if i + x in rw and j + y in rh)
for j in rh:
    for i in rw:
        if ch[j][i].isDigit():
            m += sumdir(i, j) == int(ch[j][i]) # (3)
            for k in r4:
                m += r[i][j][k] == 0 # (3)
        else:
            m += lpSum(v[i][j]) == 1 # (4)
            for x, y, k in drc:
                m += r[i][j][k] <= mx * v[i][j][k] # (5)
                if i + x in rw and j + y in rh:
                    m += r[i][j][k] <= v[i][j][k] + r[i + x][j + y][k] # (5)
                else: m += r[i][j][k] <= v[i][j][k] # (5)
%time m.solve()
for j in rh:
    for i in rw:
        if ch[j][i].isDigit():
            print(ch[j][i], end=' ')
        else:
            print('LUDR' [sum(k * int(value(v[i][j][k])) for k in r4)], end=' ')
print()
```

Wall time: 54 ms

4 R R 1 R U  
D 4 R R 2 U  
D D 2 R D 2  
1 D D 1 D U  
D 1 R D 1 U  
L L 3 R D 2

# 波及効果の解き方

データ (data/ripple.txt)

## 問題

- 各部屋のマスには1からその部屋のマス数までの数を1つずつ入れます
- 同じ数字を同じ横行、または同じ縦列に入れる場合、数字と数字の間にその数字と同じ数以上のマス目がなくてはなりません

## 変数

- v: 各位置がどの数字か (1)
- r: 各位置の数字 (2)

## 制約

- 数字があれば、その数字 (3)
- 数字は1つのみ (4)
- rをvで表現 (5)
- nマス以内に2つ以上の数字nはないこと (6)
- 各部屋内で同じ数字はないこと (7)

			3			
		2			4	
	3		4			
		2				

1	2	1	3	1	2
2	1	3	2	4	1
4	3	2	1	5	4
3	2	1	4	1	3
2	1	4	1	3	1
1	5	2	3	1	2

```
In [11]: with open('data/ripple.txt') as fp:
    nw, nh, nr = [int(s) for s in fp.readline().split(',')]
    ch = [fp.readline() for i in range(nh)]
    rms = [[eval(s.replace('-', ',')) for s in fp.readline().split(',')]] for i in
range(nr)]
    na = max(len(rm) for rm in rms)
    rw, rh, ra = range(nw), range(nh), range(na)
m = LpProblem()
v = [[[addbinvar('v') for k in ra] for j in rh] for i in rw] # (1)
r = [[addvar('r') for j in rh] for i in rw] # (2)
def dirs(i, j, k):
    for l in range(1, k + 2):
        if i + l < nw: yield v[i + l][j][k]
        if j + l < nh: yield v[i][j + l][k]
for j in rh:
    for i in rw:
        if ch[j][i].isdigit():
            m += r[i][j] == int(ch[j][i]) # (3)
            m += lpSum(v[i][j]) == 1 # (4)
            m += lpDot(ra, v[i][j]) + 1 == r[i][j] # (5)
            for k in range(1, na):
                m += lpSum(dirs(i, j, k)) <= 2 * (1 - v[i][j][k]) # (6)
for rm in rms:
    for k in range(len(rm)):
        m += lpSum(v[i][j][k] for j, i in rm) == 1 # (7)
%time m.solve()
for j in rh:
    for i in rw:
        print(int(value(r[i][j]) + 0.5), end=' ')
    print()
```

Wall time: 58 ms

```
1 2 1 3 1 2
2 1 3 2 4 1
4 3 2 1 5 4
3 2 1 4 1 3
2 1 4 1 3 2
1 5 2 3 1 1
```

# ナンバースケルトンの解き方

データ ([data/nskeleton.txt](#))

## 問題

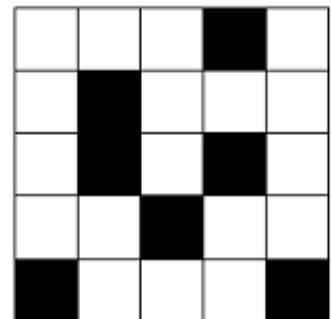
- 1つの白マスには0から9までの数字が1つだけ入ります
- 連続した白マスの数字をつなげると、与えられたナンバーの1つになります
- 各ナンバーはそのようにして一度しか出ません。
  - 異なる場所にある連続した白マスから同じナンバーは出ません
- 与えられたナンバーは全て盤面の中に見つかります
- 参考

## 変数

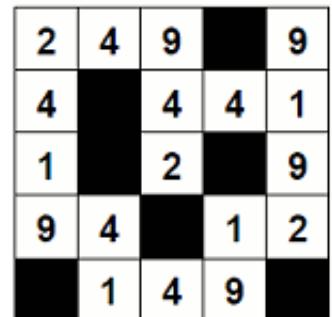
- 省略

## 制約

- 省略



12	149	2419
19	249	9192
41	441	
94	942	



```
In [12]: from collections import defaultdict
with open('data/nskeleton.txt') as fp:
    nw, nh = [int(s) for s in fp.readline().split(',')]
    rw, rh, r10 = range(nw), range(nh), range(10)
    ch = [fp.readline() for i in range(nh)]
    cands = fp.readlines()
dic = defaultdict(list)
for cand in cands:
    k = len(cand.strip())
    dic[k].append([int(cand[i]) for i in range(k)])
pos = defaultdict(list)
def addpos(i, j, x, y):
    while True:
        while i < nw and j < nh and ch[j][i] != '.':
            i, j = i + x, j + y
            if i == nw or j == nh: break
            k = 1
            while i + x * k < nw and j + y * k < nh and ch[j + y * k][i + x * k] == '.':
                k += 1
            if k > 1:
                pos[k].append([(i + x * h, j + y * h) for h in range(k)])
            i, j = i + x * k, j + y * k
for i in rw: addpos(i, 0, 0, 1)
for j in rh: addpos(0, j, 1, 0)
m = LpProblem()
v = [[[addbinvar() for k in r10] for j in rh] for i in rw]
r = [[addvar() for j in rh] for i in rw]
for j in rh:
    for i in rw:
        m += lpSum(v[i][j]) == (1 if ch[j][i] == '*' else 0)
        m += lpDot(r10, v[i][j]) == r[i][j]
for k, cnuds in dic.items():
    pss = pos[k]
    nc = len(cnuds)
    rc = range(nc)
    assert(len(pss) == nc)
    a = [[addbinvar() for g in rc] for h in rc]
    for h in rc:
        m += lpSum(a[h]) == 1
        m += lpSum(a[g][h] for g in rc) == 1
        ps = pss[h]
        for g in range(k):
            i, j = ps[g]
            for f in rc:
                m += r[i][j] <= cnuds[f][g] + 9 * (1 - a[h][f])
                m += r[i][j] >= cnuds[f][g] - 9 * (1 - a[h][f])
%time m.solve()
for j in rh:
    for i in rw:
        print('*' if ch[j][i] == '*' else '%d' % int(value(r[i][j]) + 0.5), end=' ')
    print()
```

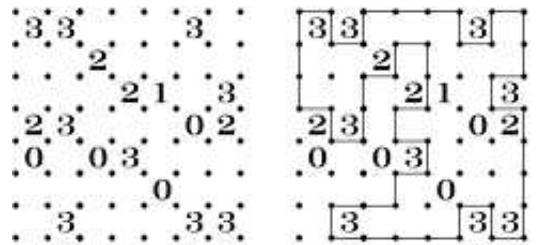
```
Wall time: 62 ms
2 4 9 * 9
4 * 4 4 1
1 * 2 * 9
9 4 * 1 2
* 1 4 9 *
```

## スリザーリンクの解き方

データ (data/slither.txt)

### 問題

- 点の間を線でつなぎ、一つの輪を作ります
- 数字は、周囲の線の数に等しいこと
- 一つの点から出る線の数は0か2となること



### 考え方

- 線は無向ですが、有向として考えます
- 数字の3がある角を始点とします(必ず通ります)

### 変数

- vh: 0:to left, 1:to right (1)
- vv: 0:to down, 1:to up (2)
- vz: 始点からの距離 (3)

### 制約

- 数字があれば、矢印の数に等しいこと (4)
- 矢印の向きは1つ (5)
- 始点以外は、矢印があれば距離が増やします (6)
- 始点の距離は0 (7)
- 距離は、全点数以下 (8)
- 入る矢印数は1以下 (9)
- 入る数と出る数は同じになること (10)

```

In [13]: with open('data/slither.txt') as fp:
    nw, nh = [int(s) for s in fp.readline().split(',')]
    M = (nw + 1) * (nh + 1) - 1
    rw, rh, rw1, rh1, r2 = range(nw), range(nh), range(nw + 1), range(nh + 1), range(2)
    ch = [fp.readline() for i in rh]
    m = LpProblem()
    vh = [[[addbinvar() for k in r2] for j in rh1] for i in rw] # 0:to left, 1:to right (1)
    vv = [[[addbinvar() for k in r2] for j in rh] for i in rw1] # 0:to down, 1:to up (2)
    vz = [[addvar() for j in rh1] for i in rw1] # (3)
    def dirs(i, j, k):
        return [vh[i - 1][j][k ^ 1] for l in r2 if 0 <= i - 1 < nw] + [
            vv[i][j - 1][k ^ 1] for l in r2 if 0 <= j - 1 < nh]
    for i in rw:
        for j in rh:
            if ch[j][i] != '.':
                m += lpSum(vh[i][j + k][l] + vv[i + k][j][l] for l in r2 for k in r2)
    == int(ch[j][i]) # (4)
            if ch[j][i] == '3': fx, fy = i, j
    m += vz[fx][fy] == 0 # (7)
    for i in rw:
        for j in rh1:
            m += lpSum(vh[i][j]) <= 1 # (5)
            m += vz[i][j] + 1 <= vz[i + 1][j] + M * (1 - vh[i][j][0]) # (6)
            if (i, j) != (fx, fy):
                m += vz[i + 1][j] + 1 <= vz[i][j] + M * (1 - vh[i][j][1]) # (6)
    for i in rw1:
        for j in rh:
            m += lpSum(vv[i][j]) <= 1 # (5)
            m += vz[i][j] + 1 <= vz[i][j + 1] + M * (1 - vv[i][j][0]) # (6)
            if (i, j) != (fx, fy):
                m += vz[i][j + 1] + 1 <= vz[i][j] + M * (1 - vv[i][j][1]) # (6)
        for j in rh1:
            m += vz[i][j] <= M # (8)
            din = dirs(i, j, 1)
            dout = dirs(i, j, 0)
            m += lpSum(din) <= 1 # (9)
            m += lpSum(din) == lpSum(dout) # (10)
    %time m.solve()
    for j in rh1:
        sys.stdout.write(' ')
        for i in rw:
            sys.stdout.write('-' if (value(vh[i][j][0]) + value(vh[i][j][1])) > 0.5 else ' ')
        sys.stdout.write('\n')
        if j == nh: break
        for i in rw1:
            sys.stdout.write('|' if (value(vv[i][j][0]) + value(vv[i][j][1])) > 0.5 else ' ')
            if i < nw: sys.stdout.write(ch[j][i])
        sys.stdout.write('\n')

```

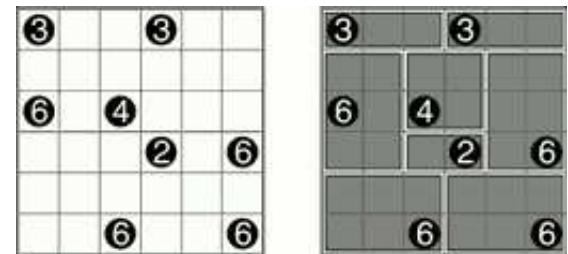
Wall time: 1.71 s

```
- - - - -  
|3|3|. . .|3|. |  
| . . 2|.|. . .|  
| . . |. 2|1 .|3  
| . . - - -  
2|3|. |. . 0 2|  
| . . - - -  
0 . 0 3|.|. . |  
| . . |. 0 . . |  
| . . - - -  
. |3 . . .|3|3|  
- - - - -
```

## 四角に切れの解き方

データ (data/square.txt)

### 問題



### 変数

- v: 各位置が各部屋かどうか (1)
- vl: 候補のどれか (2)

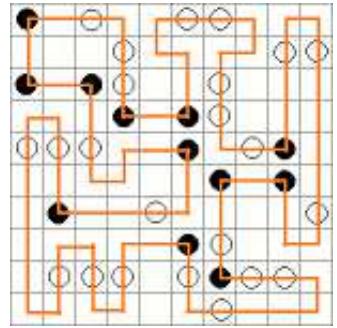
### 制約

- vは1つの部屋のみ (3)
- vlは1つの候補のみ (4)
- vlの1つを選んだら、その位置のその部屋は1 (5)

```
In [14]: with open('data/square.txt') as fp:
    nw, nh = [int(s) for s in fp.readline().split(',')]
    rw, rh = range(nw), range(nh)
    ch = [fp.readline() for j in rh]
    tgt = [(i, j, int(ch[j][i])) for j in rh for i in rw if ch[j][i].isdigit()]
    nm = len(tgt)
m = LpProblem()
v = [[[addbinvar() for k in range(nm)] for j in rh] for i in rw] # (1)
for j in rh:
    for i in rw:
        m += lpSum(v[i][j]) == 1 # (3)
def make(h, pi, pj, na):
    lst = []
    for i in range(1, na + 1):
        j = na // i
        if i * j < na: continue
        for x in range(i):
            if pi - x < 0 or pi - x + i > nw: continue
            lx = range(pi - x, pi - x + i)
            for y in range(j):
                if pj - y < 0 or pj - y + j > nh: continue
                ly = range(pj - y, pj - y + j)
                lst.append([v[dx][dy][h] for dy in ly for dx in lx])
    return lst
for h, (i, j, k) in enumerate(tgt):
    lst = make(h, i, j, k)
    v1 = [addbinvar() for l1 in lst] # (2)
    m += lpSum(v1) == 1 # (4)
    for l, l1 in enumerate(lst):
        for t in l1:
            m += v1[l] <= t # (5)
%time m.solve()
for j in rh:
    for i in rw:
        print('%d' % sum((k + 1) * value(v[i][j][k]) for k in range(nm)), end=' ')
    print()
```

Wall time: 63 ms

```
1 1 1 2 2 2
3 3 4 4 6 6
3 3 4 4 6 6
3 3 5 5 6 6
7 7 7 8 8 8
7 7 7 8 8 8
```



# ましゅの解き方

データ (data/mashu.txt)

## 問題

- 盤面に線を引き、すべての白丸と黒丸を通る1つの輪を作ります
- 線はタテヨコにマスの中央を通り、1マスに1本だけ通過できます
- 線をワクの外に出したり、交差や枝分かれさせたりしてはいけません
- 白丸を通る線は、白丸のマスで必ず直進し、白丸の両隣のマスの少なくとも片方で直角に曲がります
- 黒丸を通る線は、黒丸のマスで必ず直角に曲がりますが、黒丸の隣のマスで曲がってはいけません

## 変数

- vz: 始点からの距離 (1)
- vh: 水平線を引くかどうか (2)
- vv: 垂直線を引くかどうか (3)
- vhd: 0:to left, 1:to right (4)
- vvd: 0:to down, 1:to up (5)

## 制約

- vzはM(マス数)以下 (6)
- 点に入るのは1以下 (7)
- 点に入る数と出る数は等しくなること (8)
- の条件。●の条件 (9)
- vzの更新式。vh、vvをvhd、vvdで表現 (10)

```
In [15]: with open('data/mashu.txt') as fp:
    nw, nh = [int(s) for s in fp.readline().split(',')]
    M = nw * nh - 1
    rw, rh, rw1, rh1, r2 = range(nw), range(nh), range(nw - 1), range(nh - 1), range(2)
    ch = [fp.readline() for j in rh]
    m = LpProblem()
    vz = [[addvar() for j in rh] for i in rw] # (1)
    vh = [[addvar() for j in rh] for i in rw1] # (2)
    vv = [[addvar() for j in rh1] for i in rw] # (3)
    vhd = [[[addbinvar() for k in r2] for j in rh] for i in rw1] # 0:to left, 1:to right (4)
    vvd = [[[addbinvar() for k in r2] for j in rh1] for i in rw] # 0:to down, 1:to up (5)
    def dirs(i, j, k):
        return [vhd[i - l][j][k ^ l] for l in r2 if 0 <= i - l < nw - 1] + \
               [vvd[i][j - l][k ^ l] for l in r2 if 0 <= j - l < nh - 1]
    for i in rw:
        for j in rh:
            m += vz[i][j] <= M # (6)
            din = dirs(i, j, 1)
            dout = dirs(i, j, 0)
            m += lpSum(din) <= 1 # (7)
```

```

m += IpSum(din) == IpSum(dout) # (8)
if ch[j][i] == 'o':
    fx, fy = i, j
    m += IpSum(dirs(i, j, 1)) == 1 # (9)
    if 1 <= i < nw - 1:
        m += vh[i - 1][j] == vh[i][j] # (9)
    if 2 <= i < nw - 2:
        m += vh[i - 2][j] + vh[i + 1][j] <= 2 - vh[i][j] # (9)
    if 1 <= j < nh - 1:
        m += vv[i][j - 1] == vv[i][j] # (9)
    if 2 <= j < nh - 2:
        m += vv[i][j - 2] + vv[i][j + 1] <= 2 - vv[i][j] # (9)
elif ch[j][i] == '*':
    m += IpSum(vh[i - 1][j] for l in r2 if 0 <= i - l < nw - 1) == 1 # (9)
    if 0 <= i - 2:
        m += vh[i - 1][j] <= vh[i - 2][j] # (9)
    if i + 1 < nw - 1:
        m += vh[i][j] <= vh[i + 1][j] # (9)
    if 0 <= j - 2:
        m += vv[i][j - 1] <= vv[i][j - 2] # (9)
    if j + 1 < nh - 1:
        m += vv[i][j] <= vv[i][j + 1] # (9)
for i in rw1:
    for j in rh:
        m += IpSum(vhd[i][j]) == vh[i][j] # (10)
        m += vh[i][j] <= 1 # (10)
        m += vz[i][j] + 1 <= vz[i + 1][j] + M * (1 - vhd[i][j][0]) # (10)
        if (i, j) != (fx, fy):
            m += vz[i + 1][j] + 1 <= vz[i][j] + M * (1 - vhd[i][j][1]) # (10)
for i in rw:
    for j in rh1:
        m += IpSum(vvd[i][j]) == vv[i][j] # (10)
        m += vv[i][j] <= 1 # (10)
        m += vz[i][j] + 1 <= vz[i][j + 1] + M * (1 - vvd[i][j][0]) # (10)
        if (i, j) != (fx, fy):
            m += vz[i][j + 1] + 1 <= vz[i][j] + M * (1 - vvd[i][j][1]) # (10)
%time m.solve()
for j in rh:
    for i in rw:
        sys.stdout.write('+')
        if i < nw - 1:
            sys.stdout.write('-' if value(vh[i][j]) > 0.5 else ' ')
    sys.stdout.write('\n')
    if j == nh - 1: break
    for i in rw:
        sys.stdout.write('|' if value(vv[i][j]) > 0.5 else ' ')
    sys.stdout.write('\n')

```

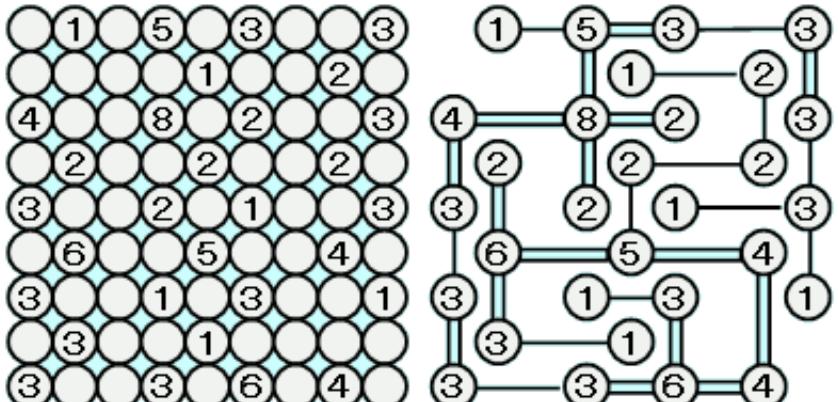
Wall time: 877 ms

## 橋をかけろの解き方

## データ (data/bridge.txt)

問題

- 島同士を線(橋)で結び、すべての数字が線でつながっているようにします
  - 線は他の線と交差したり数字を横切ったりしてはいけません
  - 線は水平方向か垂直方向に引かれます
  - どの数の間にも2本までしか線は引けません
  - 数字はその数字から引かれる線の数を表します



# 变数

- $v:0:h1, 1:h2, 2:v1, 3:v2$  (1)
  - $a:0:h, 1:v \in \{0,1,2\}$  (2)

制約

- $a$ を $v$ で表します (3)
  - 丸内が数字なら、周りの合計に等しくかつ $v$ は全て0 (4)
  - $h_2 == 1$ なら $h_1 == 1$ であること。 $v$ も同じ (5)
  - $h_1$ と $v_1$ は同時に成り立ちません (6)
  - 数字以外では線が続き、端では線ははみ出ないこと (7)
  - 全ての数字がつながること (8)

```
In [18]: from unionfind import *
with open('data/bridge.txt') as fp:
    nw, nh = [int(s) for s in fp.readline().split(',')]
    rw, rh, r2, r4 = range(nw), range(nh), range(2), range(4)
    ch = [fp.readline() for i in rh]
m = LpProblem()
v = [[[addbinvar() for k in r4] for j in rh] for i in rw] # 0:h1, 1:h2, 2:v1, 3:v2 (1)
a = [[[addvar() for k in r2] for j in rh] for i in rw] # 0:h, 1:v (2)
def dirs(i, j):
    return [a[i + x - y][j + x + y - 1][1 - x ^ y] for x in r2 for y in r2] # if i + x - y in rw and j + x + y - 1 in rh
for j in rh:
    for i in rw:
        m += a[i][j][0] == lpSum(v[i][j][:2]) # (3)
        m += a[i][j][1] == lpSum(v[i][j][2:]) # (3)
        if ch[j][i].isdigit():
            f = i + j * nw
            m += lpSum(dirs(i, j)) == int(ch[j][i]) # (4)
            for k in r4:
                m += v[i][j][k] == 0 # (4)
        else:
            m += v[i][j][0] >= v[i][j][1] # (5)
            m += v[i][j][2] >= v[i][j][3] # (5)
            m += lpSum(v[i][j][0:3:2]) <= 1 # (6)
            if i < nw - 1 and not ch[j][i + 1].isdigit():
                m += a[i][j][0] == a[i + 1][j][0] # (7)
            if j < nh - 1 and not ch[j + 1][i].isdigit():
                m += a[i][j][1] == a[i][j + 1][1] # (7)
            if i == 0 or i == nw - 1:
                m += a[i][j][0] == 0 # (7)
            if j == 0 or j == nh - 1:
                m += a[i][j][1] == 0 # (7)
while True:
    %time m.solve()
    if m.status == 1: break
    u = unionfind(nw * nh)
    e = []
    for j in rh:
        for i in rw:
            if value(a[i][j][0]) > 0.5:
                u.unite(i + j * nw, i + j * nw - 1)
                u.unite(i + j * nw - 1, i + j * nw + 1)
            elif value(a[i][j][1]) > 0.5:
                u.unite(i + j * nw, i + j * nw - nw)
                u.unite(i + j * nw - nw, i + j * nw + nw)
            else:
                e.append(lpSum(a[i][j]))
    if all([u.issame(f, i + j * nw) for i in rw for j in rh if ch[j][i].isdigit()]): break
    m += lpSum(e) >= 1 # (8)
    for j in rh:
        for i in rw:
            h = int(sum(value(v[i][j][k]) * (3 if k == 2 else 1) for k in r4))
            print(ch[j][i] if ch[j][i] != '.' else '-|H'[h], end=' ')
print()
```

```

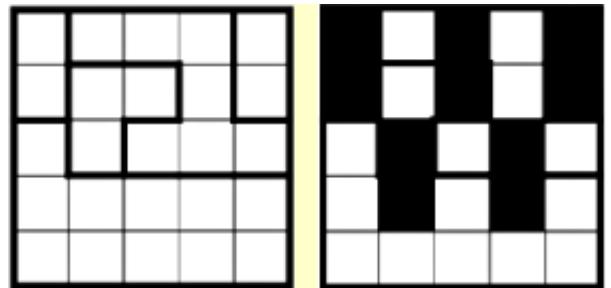
Wall time: 68 ms
1 - 5 = 3 - - 3
H 1 - - 2 H
4 = = 8 = 2 | 3
H 2   H 2 - - 2 |
3 H   2 | 1 - - 3
| 6 = = 5 = = 4 |
3 H   1 - 3   H 1
H 3 - - 1 H   H
3 - - 3 = 6 = 4

```

## のりのりの解き方

データ ([data/norinori.txt](#))

### 問題



- 盤面のいくつかのマスを黒くねります
- 黒マスは必ずタテかヨコにちょうど2つだけねります
- 太線で区切られた各部分には、黒マスが2つずつ入ります

### 変数

- v: 黒かどうか (1)

### 制約

- あるマスが白なら、周りは全て白 (2)
- あるマスが黒なら、周りは1つだけ黒 (3)
- 各部分内の黒は2つ (4)

```
In [19]: from collections import defaultdict
with open('data/norinori.txt') as fp:
    nw, nh = [int(s) for s in fp.readline().split(',')]
    rw, rh, r2 = range(nw), range(nh), range(2)
    ch = [fp.readline() for j in rh]
m = LpProblem()
v = [[addbinvar() for j in rh] for i in rw] # (1)
def dirs(i, j):
    return [v[i + x - y][j + x + y - 1] for x in r2 for y in r2 if 0 <= i + x - y < nw and 0 <= j + x + y - 1 < nh]
dic = defaultdict(list)
for j in rh:
    for i in rw:
        dic[ch[j][i]].append(v[i][j])
        m += v[i][j] <= lpSum(dirs(i, j)) # (2)
        m += 3 * v[i][j] + lpSum(dirs(i, j)) <= 4 # (3)
for l in dic.values():
    m += lpSum(l) == 2 # (4)
%time m.solve()
for j in rh:
    for i in rw:
        print('.*' [int(value(v[i][j]) + 0.5)], end=' ')
    print()
```

Wall time: 40 ms

\* . \* . \*  
\* . \* . \*  
. \* . \* .  
. \* . \* .  
. . . . .

## ブロックパズルの解き方

データ (data/block.txt)

### 問題

A	A	E	C	D	A	A	E	C	D
C	D	C	D	E	C	D	C	D	E
A	B	D	B	A	A	B	D	B	A
D	E	A	E	C	D	E	A	E	C
E	C	B	B	B	E	C	B	B	B

- 部品と部品の間に線を引いて、盤面を指定されたブロックに分けます
- ブロックの中に部品がちょうど1つ入るようにします
- それぞれのブロックに入る部品は、どれもタテかヨコにつながっているように

### 変数

- v: ブロック候補を選ぶかどうか # (1)

### 制約

- 各マスで選ばれた候補はちょうど1つ # (2)

```
In [16]: from unionfind import *
with open('data/block.txt') as fp:
    nw, nh, ns = [int(s) for s in fp.readline().split(',')]
    nb = nw * nh // ns
    rw, rh, rb = range(nw), range(nh), range(nb)
    ch = [fp.readline() for j in rh]
chk = [False] * ns
cand = []
def makecand(c, ca):
    if len(ca) == ns:
        if unionfind.isconnectedlist(nw, nh, ca): cand.append(ca)
        return
    if c == nw * nh: return
    i, j = c % nw, c // nw
    k = ord(ch[j][i]) - 65
    if not chk[k]:
        chk[k] = True
        makecand(c + 1, ca + [(i, j)])
        chk[k] = False
    makecand(c + 1, ca)
makecand(0, [])
nn = len(cand)
rn = range(nn)
m = LpProblem()
v = [addbinvar() for i in rn] # (1)
e = [[LpAffineExpression() == 1 for j in rh] for i in rw]
for i in rw:
    for j in rh:
        m += e[i][j] # (2)
for k in rn:
    ca = cand[k]
    for i, j in ca:
        e[i][j].addterm(v[k], 1)
%time m.solve()
r = [[0] * nw for j in rh]
ic = 0
for k in rn:
    if value(v[k]) < 0.5: continue
    ic += 1
    ca = cand[k]
    for i, j in ca: r[i][j] = ic
for j in rh:
    for i in rw: print(r[i][j], end=' ')
    print()
```

Wall time: 30 ms

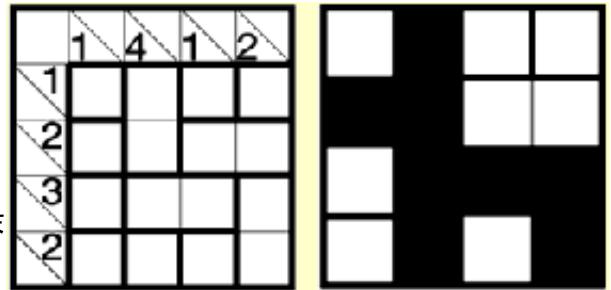
```
1 2 2 2 3
1 1 4 2 3
5 1 4 2 3
5 1 4 4 3
5 5 5 4 3
```

# タイルペイントの解き方

データ (data/tile.txt)

## 問題

- 盤面上にある、太線で区切られた部分(タイルと呼ぶ)のいくつかを黒くぬります
- 盤面の数字は、その右あるいは下の、外周か次の斜線のマスまでの区切られた一列のうちで、黒くぬられるマスの数を表します
- どのタイルも、すべてのマスをぬるかすべてのマスをぬらずにおくかのどちらかとし、タイルの一部のマスだけをぬってはいけません



## 変数

- v: 黒かどうか (1)

## 制約

- 縦及び横のヒントの数に等しい (2)
- タイルは全部塗るか塗らないか(バイナリ変数不要)(3)

```
In [17]: from collections import defaultdict
with open('data/tile.txt') as fp:
    nw, nh = [int(s) for s in fp.readline().split(',')]
    rw, rh = range(nw), range(nh)
    hw = [int(s) for s in fp.readline().split(',')]
    hh = [int(s) for s in fp.readline().split(',')]
    ch = [fp.readline() for j in rh]
    m = LpProblem()
    v = [[[addbinvar() for j in rh] for i in rw] # (1)
          for i in rw:
              m += lpSum(v[i]) == hw[i] # (2)
          for j in rh:
              m += lpSum(v[i][j] for i in rw) == hh[j] # (2)
          dic = defaultdict(list)
          for i in rw:
              for j in rh: dic[ch[j][i]].append(v[i][j])
          for d in dic.values():
              for vi, vj in zip(d, d[1:]):
                  m += vi == vj # (3)
    %time m.solve()
    for j in rh:
        for i in rw:
            print('.#' [int(value(v[i][j]) + 0.5)], end=' ')
    print()
```

Wall time: 80 ms

```
. # .
# # .
. # #
. # . #
```

# 因子の部屋の解き方

データ (data/inshi.txt)

## 問題

6	15		1	12
	20		8	
10		6		
4	4		15	
	1		10	

2	3	5	1	4
3	5	4	2	1
5	2	1	4	3
1	4	2	3	5
4	1	3	5	2

- すべてのマスに1からNまでの数字のどれかを1つずつ入ります(0は使いません)
- タテ列、ヨコ列のどれにも、1からNまでの数字が1つずつ入ります
- 太線で囲まれた四角形(部屋)の左上のマスに小さく書かれている数は、その部屋の各マスに入る数をすべてかけあわせた値となります

## 変数

- v:各マスでその数字かどうか (1)
- r:マスの数字 (2)

## 制約

- $v_{ij}$ は1つの数字のみ (3)
- rをvで表す (4)
- 縦および横で同じ数字はない (5)
- 積が指定した数字 (6)

```
In [18]: from collections import defaultdict
from math import log
with open('data/inshi.txt') as fp:
    nn, nb, nm = [int(s) for s in fp.readline().split(',')]
    rn, rb, rm = range(nn), range(nb), range(nm)
    ch = [fp.readline() for j in rn]
    ns = [log(int(fp.readline())) for l in rb]
m = LpProblem()
v = [[[addbinvar() for k in rm] for j in rn] for i in rn] # (1)
r = [[addvar() for j in rn] for i in rn] # (2)
dic = defaultdict(list)
for i in rn:
    for j in rn:
        m += lpSum(v[i][j]) == 1 # (3)
        m += lpDot(rm, v[i][j]) + 1 == r[i][j] # (4)
        dic[ch[j][i]].append((i, j))
    for k in rm:
        m += lpSum(v[i][j][k] for j in rn) == 1 # (5)
for j in rn:
    for k in rm:
        m += lpSum(v[i][j][k] for i in rn) == 1 # (5)
for l in rb:
    s = [log(k + 1) * v[i][j][k] for i, j in dic[chr(l + 65)] for k in rm]
    m += lpSum(s) >= ns[l] - 0.0001 # (6)
    m += lpSum(s) <= ns[l] + 0.0001 # (6)
%time m.solve()
for j in rn:
    for i in rn:
        print(int(value(r[i][j]) + 0.5), end=' ')
    print()
```

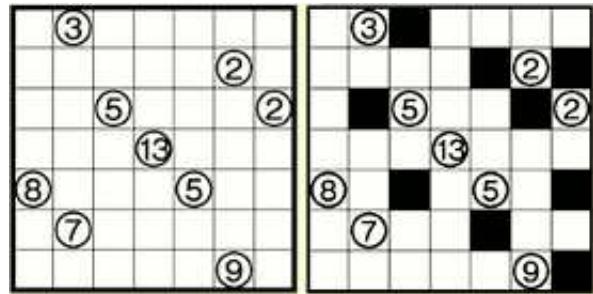
Wall time: 52 ms

2 3 5 1 4  
 3 5 4 2 1  
 5 2 1 4 3  
 1 4 2 3 5  
 4 1 3 5 2

# 黒どこの解き方

データ ([data/kurodoko.txt](#))

## 問題



- 盤面のいくつかのマスを黒くねります
- 盤面の数字は、その数字から上下左右4方向にまっすぐ進み、次の黒マスか外周にたどりつくまでの、その数字を含めてのマス数の合計を表します
- 数字が入っているマスを黒くねってはいけません
- 黒マスをタテヨコに連続させたり、黒マスで盤面を分断したりしてはいけません

## 変数

- vb: 0: white, 1: black (1)
- vd: 0: left, 1: up, 2: right, 3:down (2)

## 制約

- 各マスが黒ならvdは全方向とも0 (3)
- 各マスが白ならvdは方向先のvdより1大きいこと (4)
- 端のvdは、白なら1、黒なら0 (5)
- 黒は連続しないこと (6)
- 数字ならマスは白 (7)
- 数字なら $vd_{ij}$ の和は数字+3に等しいこと (8)
- 黒マスが分断しないこと (9)

```
In [19]: from unionfind import *
with open('data/kurodoko.txt') as fp:
    nw, nh = [int(s) for s in fp.readline().split(',')]
    rw, rh, r4 = range(nw), range(nh), range(4)
    ch = [fp.readline() for i in rh]
m = LpProblem()
vb = [[addbinvar() for j in rh] for i in rw] # 0: white, 1: black (1)
vd = [[[addvar() for k in r4] for j in rh] for i in rw] # 0: left, 1: up, 2: right, 3:down (2)
for i in rw:
    for j in rh:
        for k in r4:
            mx = nw if k % 2 == 0 else nh
            m += vd[i][j][k] <= (1 - vb[i][j]) * mx # (3)
            ik, jk = i + [-1, 0, 1, 0][k], j + [0, -1, 0, 1][k]
            if 0 <= ik < nw and 0 <= jk < nh:
                m += vd[i][j][k] >= vd[ik][jk][k] + 1 - mx * vb[i][j] # (4)
                m += vd[i][j][k] <= vd[ik][jk][k] + 1 + mx * vb[i][j] # (4)
            else:
                m += vd[i][j][k] == 1 - vb[i][j] # (5)
            if i > 0: m += vb[i - 1][j] + vb[i][j] <= 1 # (6)
            if j > 0: m += vb[i][j - 1] + vb[i][j] <= 1 # (6)
            if ch[j][i] != '.':
                m += vb[i][j] == 0 # (7)

            n = int(ch[j][i]) if ch[j][i].isdigit() else ord(ch[j][i]) - 87
            m += lpSum(vd[i][j]) == n + 3 # (8)

while True:
    %time m.solve()
    if unionfind.isconnected([[value(vb[i][j]) < 0.5 for j in rh] for i in rw]):
        break
    s = [vb[i][j] for i in rw for j in rh if value(vb[i][j]) > 0.5]
    m += lpSum(s) <= len(s) - 1 # (9)
for j in rh:
    for i in rw:
        print(ch[j][i] if ch[j][i] != '.' else '#' if value(vb[i][j]) > 0.5 else '.', end=' ')
    print()
```

Wall time: 123 ms

```
. 3 # . . .
. . . . # 2 #
. # 5 . . # 2
. . . d . .
8 . # . 5 . #
. 7 . . # . .
. . . . . 9 #
```

# 推理パズルの解き方

データ (data/suiri.txt)

## 問題

- 3つの組を入力し、各組間の対応を求めます
- 入力ファイルの意味
  - ヒント数
  - 組1のリスト
  - 組2のリスト
  - 組3のリスト
  - 以降ヒント
    - 明は白いのを買った
    - 明は糸じゃない
    - 青い紙を買った人がいる
    - 勇は紙じゃない
    - 正は靴を買った
    - 正は赤じゃない

	拿	靴	紙	糸	赤	青	白	黒
明	○	×	×	×	×	×	○	×
勇	×	×	×	○	○	×	×	×
正	×	○	×	×	×	×	×	○
洋	×	×	○	×	×	○	×	×
赤	×	×	×	○				
青	×	×	○	×				
白	○	×	×	×				
黒	×	○	×	×				

## 変数

- v12 (1)
- v23 (2)
- v31 (3)

## 制約

- 縦横で丸は1つ (4)
- AかつB、BかつCなら、CかつA (5)
- ヒントを満たすこと (6)

```
In [20]: with open('data/suiri.txt') as fp:
    nh = int(fp.readline())
    rh, r3, r4 = range(nh), range(3), range(4)
    it = [fp.readline().rstrip('\n').split(',') for i in r3]
    hh = [fp.readline().rstrip('\n').split(',') for h in rh]
    dic = [[] for i in r3]
    for i in r3:
        for j in r4:
            dic[i][it[i][j]] = j
    m = LpProblem()
    v12 = [[addbinvar() for j in r4] for i in r4] # (1)
    v23 = [[addbinvar() for j in r4] for i in r4] # (2)
    v31 = [[addbinvar() for j in r4] for i in r4] # (3)
    for i in r4:
        m += IpSum(v12[i]) == 1 # (4)
        m += IpSum(v12[j][i] for j in r4) == 1 # (4)
        m += IpSum(v23[i]) == 1 # (4)
        m += IpSum(v23[j][i] for j in r4) == 1 # (4)
        m += IpSum(v31[i]) == 1 # (4)
        m += IpSum(v31[j][i] for j in r4) == 1 # (4)
        for j in r4:
            for k in r4:
                m += v12[i][j] + v23[j][k] - v31[k][i] <= 1 # (5)
                m += v23[i][j] + v31[j][k] - v12[k][i] <= 1 # (5)
                m += v31[i][j] + v12[j][k] - v23[k][i] <= 1 # (5)
    for h in rh:
        c0, c1, c2, c3 = [int(eval(hh[h][0]))] + [dic[i].get(hh[h][i + 1], -1) for i in r3]
        if c1 >= 0 and c2 >= 0: m += v12[c1][c2] == c0 # (6)
        if c2 >= 0 and c3 >= 0: m += v23[c2][c3] == c0 # (6)
        if c3 >= 0 and c1 >= 0: m += v31[c3][c1] == c0 # (6)
    %time m.solve()
    for i in r4:
        print(it[0][i], end=' ')
        j = [j for j in r4 if value(v12[i][j]) > 0.5][0]
        print(it[1][j], end=' ')
        k = [k for k in r4 if value(v23[j][k]) > 0.5][0]
        print(it[2][k])
```

Wall time: 92 ms  
akira umbrella white  
isamu string red  
tadashi shoes black  
hiroshi paper blue

1	8	6	2	6	7	5	3
3	1	1	1	8	2	2	2
8	3	2	4	7	6	5	1
3	7	5	8	3	3	1	4
5	4	4	6	7	1	8	2
7	1	4	3	2	5	3	5
2	2	8	3	4	4	7	5
2	2	3	1	4	4	6	5

# ひとりにしてくれの解き方

データ (data/hitori.txt)

## 問題

- 盤面に並んでいる数字のうちいくつかを黒くぬり、タテでもヨコでも同じ列に同じ数字が複数個入らないようにします
- 黒マスをタテヨコに連続したり、黒マスで盤面を分断してはいけません

## 変数

- v:0:number, 1:black (1)

## 制約

- 黒以外の数字は縦横に重複しないこと (2)
- 黒は連続しないこと (3)
- 黒で分断しないこと (4)

```
In [21]: from unionfind import *
with open('data/hitori.txt') as fp:
    nn = int(fp.readline())
    rn = range(nn)
    ch = [fp.readline() for j in rn]
m = LpProblem()
v = [[addbinvar() for j in rn] for i in rn] # 0:number, 1:black (1)
for i in rn:
    for j in rn:
        if i > 0: m += v[i - 1][j] + v[i][j] <= 1 # (2)
        if j > 0: m += v[i][j - 1] + v[i][j] <= 1 # (2)
        s = [v[i][k] for k in rn if int(ch[k][i]) == j + 1]
        if len(s) > 1: m += lpSum(s) >= len(s) - 1 # (3)
        s = [v[k][i] for k in rn if int(ch[i][k]) == j + 1]
        if len(s) > 1: m += lpSum(s) >= len(s) - 1 # (3)
while True:
    %time m.solve()
    if unionfind.isconnected([[value(v[i][j]) < 0.5 for i in rn] for j in rn]): break
    s = [v[i][j] for i in rn for j in rn if value(v[i][j]) > 0.5]
    m += lpSum(s) <= len(s) - 1 # (34)
for j in rn:
    for i in rn:
        print('#' if value(v[i][j]) > 0.5 else ch[j][i], end=' ')
    print()
```

```
Wall time: 76 ms
Wall time: 37 ms
Wall time: 37 ms
Wall time: 51 ms
Wall time: 64 ms
Wall time: 52 ms
Wall time: 59 ms
Wall time: 60 ms

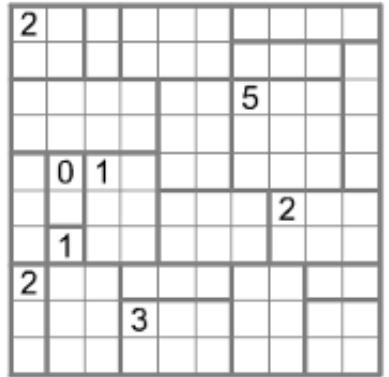
Wall time: 54 ms
Wall time: 59 ms
Wall time: 59 ms
Wall time: 54 ms
Wall time: 62 ms
Wall time: 59 ms
Wall time: 188 ms
Wall time: 88 ms
Wall time: 75 ms
Wall time: 57 ms
Wall time: 62 ms
Wall time: 70 ms
Wall time: 65 ms
Wall time: 59 ms
Wall time: 63 ms
Wall time: 64 ms
Wall time: 59 ms
Wall time: 59 ms
Wall time: 62 ms
Wall time: 63 ms
Wall time: 64 ms
Wall time: 65 ms
Wall time: 69 ms
Wall time: 81 ms
Wall time: 67 ms
Wall time: 74 ms
Wall time: 75 ms
1 8 # 2 6 7 5 3
3 # 1 # 8 # 2 #
8 3 2 4 7 6 # 1
# 7 5 8 3 # 1 4
5 4 # 6 # 1 8 2
7 1 4 # 2 5 3 #
2 # 8 3 4 # 7 5
# 2 3 1 # 4 6 #
```

# へやわけの解き方

データ ([data/heyawake.txt](#))

## 問題

- 盤面のいくつかのマスを黒くねります
- 太線で区切られた四角(部屋)に入っている数字は、その部屋に入る黒マスの数を表します
- 数字の入っていない部屋は、いくつ黒マスが入るか不明です
- 白マスを、タテまたはヨコにまっすぐに3つ以上の部屋にわたって続けさせてはいけません
- 黒マスをタテヨコに連続させたり、黒マスで盤面を分断したりしてはいけません



## 変数

- v: 0:white, 1:black (1)

## 制約

- 3つの部屋で白をまっすぐ連続してはいけません (2)
- 数字は部屋内の黒の数となること (3)
- 黒は連続しないこと (4)
- 黒で分断しないこと (5)

```

In [22]: from unionfind import *
from collections import defaultdict
with open('data/heyawake.txt') as fp:
    nw, nh, nn = [int(s) for s in fp.readline().split(',')]
    rw, rh, rn, r2 = range(nw), range(nh), range(nn), range(2)
    ch = [fp.readline() for j in rh]
    hh = [fp.readline().split(',') for h in rn]
dic = {}
for i in rw:
    for j in rh:
        it = dic.get(ch[j][i], [i, j, i, j])
        dic[ch[j][i]] = [min(it[0], i), min(it[1], j), max(it[0], i), max(it[1], j)]
m = LpProblem()
v = [[addbinvar() for j in rh] for i in rw] # 0:white, 1:black (1)
for i in rw:
    for j in rh:
        if i > 0: m += v[i - 1][j] + v[i][j] <= 1 # (4)
        if j > 0: m += v[i][j - 1] + v[i][j] <= 1 # (4)
for c, s in hh:
    mni, mnj, mxi, mxj = dic[c]
    ni, nj = mxi - mni + 1, mxj - mnj + 1
    m += lpSum(v[i + mni][j + mnj] for i in range(ni) for j in range(nj)) == int(s) # (3)
for mni, mnj, mxi, mxj in dic.values():
    ni, nj = mxi - mni + 1, mxj - mnj + 1
    if mnj > 0 and mxj < nh - 1:
        for i in range(ni):
            m += lpSum(v[i + mni][j] for j in range(mnj - 1, mxj + 2)) >= 1 # (2)
    if mni > 0 and mxi < nw - 1:
        for j in range(nj):
            m += lpSum(v[i][j + mnj] for i in range(mni - 1, mxi + 2)) >= 1 # (2)
def dirs(i, j):
    return [i + x - y + nw * (j + x + y - 1) for x in r2 for y in r2] #
        if 0 <= i + x - y < nw and 0 <= j + x + y - 1 < nh]
while True:
    %time m.solve()
    t = [[value(v[i][j]) < 0.5 for j in rh] for i in rw]
    u = unionfind(nw * nh)
    if unionfind.isconnected(t, u): break
    dc = defaultdict(list)
    for i in rw:
        for j in rh:
            if t[i][j]: continue
            lI = list(set(u.find(k) for k in dirs(i, j)))
            if len(lI) >= 2:
                for l in lI: dc[l].append(v[i][j])
    for s in dc.values():
        m += lpSum(s) <= len(s) - 1 # (5)
for j in rh:
    for i in rw:
        print('#' if value(v[i][j]) > 0.5 else ch[j][i], end=' ')
print()

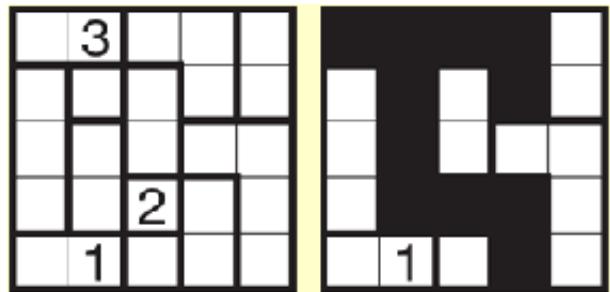
```

```
Wall time: 60 ms
Wall time: 68 ms
Wall time: 73 ms
Wall time: 91 ms
# A B # C C D # D #
A # B C C # E E E F
G G # G H H # I # F
G G G # H H I # I F
# K L L # H # I # F
J K # L M # M N N N
J # L L # M M # N #
# Q Q R R R # S T T
P Q Q # U # S S # V
# Q Q U # U S # V V
```

## ペイントエリアの解き方

データ (data/paint.txt)

### 問題



- 盤面上にある、太線で区切られた部分(タイル)のいくつかを黒くぬります
- 盤面の数字は、その数字の入っているマスにタテヨコに隣り合うマスのうち、黒マスになるマスの数を表します
- 数字のマスが黒マスになることもあります
- どのタイルも、すべてのマスをぬるかすべてのマスをぬらずにおくかのどちらとし、タイルの一部のマスだけをぬってはいけません
- すべての黒マスはつながること
- 黑白マスとも、 $2 \times 2$ 以上はだめ

### 変数

- v: 0:white, 1:black (1)

### 制約

- $2 \times 2$ の黒がないこと (2)
- タイル内は同じこと (3)
- 数字は周りの黒の数と等しいこと (4)
- 全ての黒がつながること (5)

```
In [23]: from unionfind import *
from collections import defaultdict
with open('data/paint.txt') as fp:
    nw, nh, nn = [int(s) for s in fp.readline().split(',')]
    rw, rh, rn, r2 = range(nw), range(nh), range(nn), range(2)
    ch = [fp.readline() for j in rh]
    hh = [[int(s) for s in fp.readline().split(',')] for h in rn]
m = LpProblem()
v = [[addbinvar() for j in rh] for i in rw] # 0:white, 1:black (1)
dic = defaultdict(list)
for i in rw:
    for j in rh:
        dic[ch[j][i]].append(v[i][j])
        if i > 0 and j > 0:
            m += lpSum(v[i - x][j - y] for x in r2 for y in r2) <= 3 # (2)
for ss in dic.values():
    for s, t in zip(ss, ss[1:]): m += s == t # (3)
def dirs(i, j):
    return [v[i + x - y][j + x + y - 1] for x in r2 for y in r2] #
    if 0 <= i + x - y < nw and 0 <= j + x + y - 1 < nh]
for i, j, k in hh:
    m += lpSum(dirs(i, j)) == k # (4)
while True:
    %time m.solve()
    if unionfind.isconnected([[value(v[i][j]) > 0.5 for i in rw] for j in rh]): break
    m += lpSum([v[i][j] for i in rw for j in rh if value(v[i][j]) < 0.5]) >= 1 # (5)
for j in rh:
    for i in rw:
        print('#' if value(v[i][j]) > 0.5 else ch[j][i], end=' ')
    print()
```

Wall time: 30 ms

Wall time: 27 ms

# # # # C

D # F # C

D # F H H

D # # # H

K K L # H

# 数コロの解き方

データ ([data/suukoro.txt](#))

## 問題

1	1			1
1	3	3	2	
2	4	4		
3	1	1	3	
1		1		

1	1	2		1
3	1	3	2	3
2		2		2
3	2	3	4	2
2	2	3		2
3	1	1	3	1
1		1	2	

- 全てのマスを1から4の数字か空白にします
- 数字は、そのマスの隣接マスに数字が入るマスの数になります
- 同じ数字を連続してはいけません
- すべての数字を連結すること

## 変数

- v:0:white, 1-4:number (1)
- r:数字 (2)

## 制約

- 数字があればその数字 (3)
- 数字は1つ (4)
- rをvで表現 (5)
- 数字は周りの数字の数に等しいこと (6)
- 同じ数字は連続しないこと (7)
- 全ての数字がつながること (8)

```
In [28]: from unionfind import *
with open('data/suukoro.txt') as fp:
    nw, nh = [int(s) for s in fp.readline().split(',')]
    rw, rh, r2, r5 = range(nw), range(nh), range(2), range(5)
    ch = [fp.readline() for j in rh]
m = LpProblem()
v = [[[addbinvar() for k in r5] for j in rh] for i in rw] # 0:white, 1-4:number
r = [[addvar() for j in rh] for i in rw] # (2)
def dirs(i, j):
    return [1 - v[i + x - y][j + x + y - 1][0] for x in r2 for y in r2] # (1)
    if 0 <= i + x - y < nw and 0 <= j + x + y - 1 < nh]
for i in rw:
    for j in rh:
        if ch[j][i].isdigit():
            m += v[i][j][int(ch[j][i])] == 1 # (3)
            m += lpSum(v[i][j]) == 1 # (4)
            m += lpDot(r5, v[i][j]) == r[i][j] # (5)
            m += lpSum(dirs(i, j)) >= r[i][j] # (6)
            m += lpSum(dirs(i, j)) <= r[i][j] + 4 * v[i][j][0] # (6)
            for k in range(1, 5):
                if i < nw - 1:
                    m += v[i][j][k] + v[i + 1][j][k] <= 1 # (7)
                if j < nh - 1:
                    m += v[i][j][k] + v[i][j + 1][k] <= 1 # (7)
while True:
    %time m.solve()
    if unionfind.isconnected([[value(v[i][j][0]) < 0.5 for i in rw] for j in rh]):
        break
    s = [v[i][j][0] for i in rw for j in rh if value(v[i][j][0]) > 0.5]
    m += lpSum(s) <= len(s) - 1 # (8)
for j in rh:
    for i in rw:
        print('.1234'[int(value(r[i][j]))], end=' ')
    print()
```

Wall time: 78 ms

Wall time: 77 ms

1 . 1 2 . . 1

3 1 . 3 2 3 2

2 . . 2 . 2 .

3 2 3 4 2 4 1

2 . 2 3 . 2 .

3 1 . 1 . 3 1

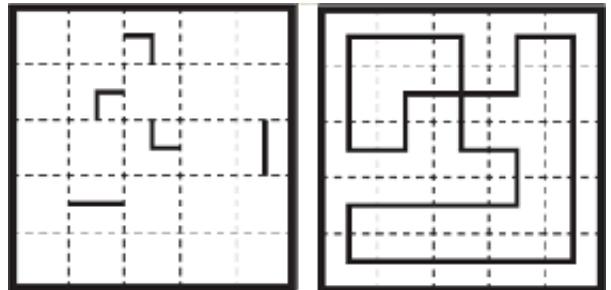
1 . . . 1 2 .

# パイプリンクの解き方

データ (data/pipalink.txt)

## 問題

- すべてのマスに線をひき1つのリンクにします
- 線は交差してもよいが、枝分かれはしません
- 最初に線が引いてあるマスは形を変えてはいけません



## 変数

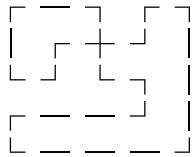
- vh (1)
- vv (2)
- vr: 表示用 (3)

## 制約

- 指定した形はそのままとすること (4)
- 全マスに線を引くこと (5)
- 交差はいいが、枝分かれしないこと (6)
- 1つのリンクにすること (7)
- 外にはみ出ないこと (8)
- vrをvhとvvで表現 (9)

```
In [27]: from unionfind import *
with open('data/pipelink.txt') as fp:
    nw, nh = [int(s) for s in fp.readline().split(',')]
    rw, rh, rw1, rh1, r2, r4 = range(nw), range(nh), range(nw + 1), range(nh + 1),
    range(2), range(4)
    ch = [fp.readline() for j in range(2 * nh - 1)]
m = LpProblem()
vh = [[addbinvar() for j in rh] for i in rw1] # (1)
vv = [[addbinvar() for j in rh1] for i in rw] # (2)
vr = [[addvar() for j in rh] for i in rw] # (3)
for j in rh:
    m += vh[0][j] == 0 # (8)
    m += vh[nw][j] == 0 # (8)
for i in rw:
    m += vv[i][0] == 0 # (8)
    m += vv[i][nh] == 0 # (8)
    for j in rh:
        if i > 0 and ch[2 * j][2 * i - 1] == '-': m += vh[i][j] == 1 # (4)
        if j > 0 and ch[2 * j - 1][2 * i] == '|': m += vv[i][j] == 1 # (4)
        l = [vv[i][j], vh[i][j], vv[i][j + 1], vh[i + 1][j]]
        m += lpSum(l) >= 2 # (5)
        for k in r4: m += lpDot([-1 if p == k else 1 for p in r4], l) <= 2 # (6)
        m += lpDot([2**k for k in r4], l) == vr[i][j] # (9)
while True:
    %time m.solve()
    if m.status != 1: break
    b = [[value(vr[i][j]) > 14.5 for j in rh] for i in rw]
    u = unionfind(nw * nh)
    for i in rw:
        for j in rh:
            if b[i][j]:
                u.unite(i - 1 + j * nw, i + 1 + j * nw)
                u.unite(i + j * nw - nw, i + j * nw + nw)
            else:
                if value(vh[i][j]) > 0.5 and (i == 0 or not b[i - 1][j]):
                    u.unite(i + j * nw, i - 1 + j * nw)
                if value(vv[i][j]) > 0.5 and (j == 0 or not b[i][j - 1]):
                    u.unite(i + j * nw, i + j * nw - nw)
    if all([b[i][j] or u.issame(0, i + j * nw) for i in rw for j in rh]): break
    for gr in u.groups():
        if len(gr) == 1: continue
        l = []
        for k in gr:
            i, j = k % nw, k // nw
            l.extend(vh[i + p][j] for p in r2 if value(vh[i + p][j]) > 0.5)
            l.extend(vv[i][j + p] for p in r2 if value(vv[i][j + p]) > 0.5)
        m += lpSum(l) <= len(l) - 2 # (7)
for j in rh:
    for i in rw:
        print(u'012└ 4 | ┐ 78 ┌—1 └34+[' + str(int(value(vr[i][j]))) +'], end=' ')
print()
```

```
Wall time: 49 ms  
Wall time: 44 ms
```



## クリークの解き方

データ ([data/kuriku.txt](#))

### 問題

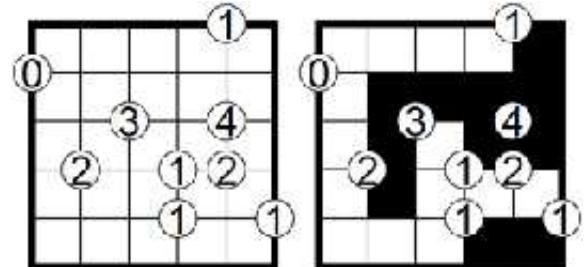
- いくつかのマスを黒くぬります
- 数字は、数字が隣接するマス中の黒マスの数を表します
- すべての白マスは連結すること

### 変数

- v: 0:white, 1:black (1)

### 制約

- 数字と黒マス数が等しいこと (2)
- 全白マスが連結すること (3)



```
In [28]: from unionfind import *
with open('data/kuriku.txt') as fp:
    nw, nh = [int(s) for s in fp.readline().split(',')]
    rw, rh, rw1, rh1, r2 = range(nw), range(nh), range(nw + 1), range(nh + 1), range(2)
    ch = [fp.readline() for j in rh1]
m = LpProblem()
v = [[addbinvar() for j in rh] for i in rw] # 0:white, 1:black (1)
for i in rw1:
    for j in rh1:
        if ch[j][i].isdigit():
            m += lpSum(v[i - x][j - y] for x in r2 if 0 <= i - x < nw !=
                        for y in r2 if 0 <= j - y < nh) == int(ch[j][i]) # (2)
while True:
    %time m.solve()
    if unionfind.isconnected([[value(v[i][j]) is None or value(v[i][j]) < 0.5 for
i in rw] for j in rh]): break
    s = [v[i][j] for i in rw for j in rh if value(v[i][j]) and value(v[i][j]) > 0.5]
    m += lpSum(s) <= len(s) - 1 # (3)
for j in rh:
    for i in rw:
        print('.' if value(v[i][j]) is None or value(v[i][j]) < 0.5 else '#', end=' ')
    print()
```

Wall time: 28 ms

Wall time: 24 ms

Wall time: 46 ms

Wall time: 31 ms

. . . . #

. # # # #

. # . # #

. # . . .

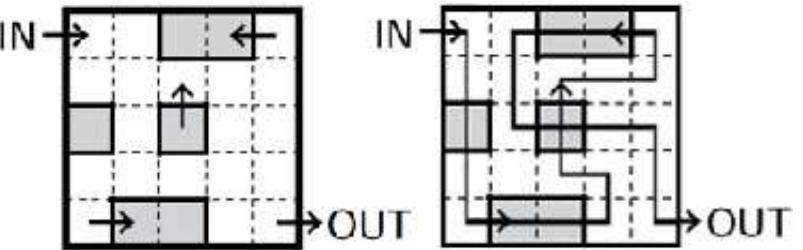
. . . # #

# アイスバーンの解き方

データ (data/eisbahn.txt)

## 問題

- INからOUTにいく1本の線をひきます
- 灰色のマスをアイスバーンとし、必ず通ります
- アイスバーンで曲がってはいけません
- アイスバーンのみ交差可です
- 矢印は必ず通ること



## 変数

- vh:0:L, 1:R (1)
- vv:0:U, 1:D (2)
- vhs (3)
- vvs (4)

## 制約

- vhsをvhで、vvsをvvで表現 (5)
- 矢印は必ず通ること (6)
- 各マスで入る数と出る数が同じこと (7)
- アイスバーンなら横は同じ、縦も同じこと(曲がらない) (8)
- アイスバーンなら線は2以上 (9)
- アイスバーンでないなら線は2以下 (10)
- 線は1つ (11)

```
In [31]: from unionfind import *
with open('data/eisbahn.txt') as fp:
    nw, nh = [int(s) for s in fp.readline().split(',')]
    rw, rh, rw1, rh1, r2, r12 = range(nw), range(nh), range(nw + 1), range(nh + 1),
    range(2), range(1, 3)
    ch = [fp.readline() for j in range(2 * nh + 1)]
m = LpProblem()
vh = [[[addbinvar() for k in r2] for j in rh] for i in rw1] # 0:L, 1:R (1)
vv = [[[addbinvar() for k in r2] for j in rh1] for i in rw] # 0:U, 1:D (2)
vhs = [[addvar() for j in rh] for i in rw1] # (3)
vvs = [[addvar() for j in rh1] for i in rw] # (4)
for j in rh1:
    for i in rw:
        c = ch[j * 2][i * 2 + 1]
        m += lpDot(r12, vv[i][j]) == vvs[i][j] # (5)
        m += vvs[i][j] <= (0 if c == '.' and j % nh == 0 else 2) # (6)
        if c == 'U': m += vv[i][j][1] + 1 <= vv[i][j][0] # (6)
        elif c == 'D': m += vv[i][j][0] + 1 <= vv[i][j][1] # (6)
        if j == nh: break
    for i in rw1:
        c = ch[j * 2 + 1][i * 2]
        m += lpDot(r12, vh[i][j]) == vhs[i][j] # (5)
```

```

m += vhs[i][j] <= (0 if c == '.' and i % nw == 0 else 2) # (6)
if c == 'L': m += vh[i][j][1] + 1 <= vh[i][j][0] # (6)
elif c == 'R': m += vh[i][j][0] + 1 <= vh[i][j][1] # (6)
for i in rw:
    e1 = lpSum(vv[i][j+k][1-k] + vh[i+k][j][1-k] for k in r2)
    e2 = lpSum(vv[i][j+k][k] + vh[i+k][j][k] for k in r2)
    m += e1 == e2 # (7)
    if ch[j*2+1][i*2+1] == '*':
        m += vhs[i][j] == vhs[i+1][j] # (8)
        m += vvs[i][j] == vvs[i][j+1] # (8)
        m += e1 + e2 >= 2 # (9)
    else: m += e1 + e2 <= 2 # (10)
while True:
    %time m.solve()
    if m.status != 1: break
    b = [[all([value(vhs[i+k][j]) > 0.5 and value(vvs[i][j+k]) > 0.5 for k in r2]) for j in rh] for i in rw]
    e = [[all([value(vhs[i+k][j]) < 0.5 and value(vvs[i][j+k]) < 0.5 for k in r2]) for j in rh] for i in rw]
    u = unionfind(nw * nh)
    p = -1
    for i in rw:
        for j in rh:
            if not e[i][j]: p = i + j * nw
            if b[i][j]:
                u.unite(i - 1 + j * nw, i + 1 + j * nw)
                u.unite(i + j * nw - nw, i + j * nw + nw)
            else:
                if i > 0 and value(vhs[i][j]) > 0.5 and not b[i-1][j]:
                    u.unite(i + j * nw, i - 1 + j * nw)
                if j > 0 and value(vvs[i][j]) > 0.5 and not b[i][j-1]:
                    u.unite(i + j * nw, i + j * nw - nw)
    if all([b[i][j] or e[i][j] or u.issame(p, i + j * nw) for i in rw for j in rh]): break
    for gr in u.groups():
        if len(gr) == 1: continue
        s = []
        for g in gr:
            i, j = g % nw, g // nw
            for k in r2:
                for l in r2:
                    if value(vh[i+k][j][l]) > 0.5: s.append(vh[i+k][j][l])
                    if value(vv[i][j+k][l]) > 0.5: s.append(vv[i][j+k][l])
        m += lpSum(s) <= len(s) - 2 # (11)
for j in rh1:
    for i in rw:
        sys.stdout.write(' %c' % '.UD'[int(value(vvs[i][j]))])
    sys.stdout.write('¥n')
    if j == nh: break
    for i in rw1:
        sys.stdout.write('%c%c' % ('.LR'[int(value(vhs[i][j]))],
                                    '¥n' if i == nw else ch[j*2+1][i*2+1]))

```

```
Wall time: 47 ms
Wall time: 56 ms
Wall time: 72 ms
```

```
.
R . L*L*L .
D D . . U
. . R R .
D D U . .
.* R*R R .
D . U . D
. . . L . .
D . . U D
. R*R*R . R
.
.
```

## サムラインの解き方

データ ([data/sumline.txt](#))

## 問題

	A	B	C	D	E	F	$\Theta$	$\Downarrow$	
a							1687	266	1 6 7 8 9
b							8361	597	7 8 3 5 4
c							465	824	4 5 9 6
d							721	829	9 7 1 2
e							9139	570	9 1 3 4 5
f							5700	352	2 5 6 9 8

- 1から9の数字をいれます

- タテヨコ各列のカギは、その列に入る数の合計です
- タテもヨコも、同じ数字は1つまでです

## 変数

- v (1)
- r (2)

## 制約

- $v_{ij}$  は1つ (3)
- rをvで表現 (4)
- 縦も横も同じ数字は1つまで (5)
- ヒントを満たすこと (6)

```
In [29]: with open('data/sumline.txt') as fp:
    nw, nh = [int(s) for s in fp.readline().split(',')]
    rw, rh, r9 = range(nw), range(nh), range(9)
    ch = [fp.readline() for j in rh]
    hh = [int(fp.readline()) for j in rh]
    hv = [int(fp.readline()) for i in rw]
m = LpProblem()
v = [[[addbinvar() for k in r9] for j in rh] for i in rw]
r = [[addvar() for j in rh] for i in rw]
def addsum(i, j, x, y):
    e, f = LpAffineExpression(), LpAffineExpression()
    while i < nw and j < nh:
        if ch[j][i] == '.':
            f = f * 10 + r[i][j]
        else:
            e += f
            f = LpAffineExpression()
            i, j = i + x, j + y
    return e + f
for i in rw:
    for j in rh:
        m += lpSum(v[i][j]) == 1
        m += lpDot(r9, v[i][j]) + 1 == r[i][j]
    for k in r9:
        m += lpSum(v[i][j][k] for j in rh) <= 1
    m += addsum(i, 0, 0, 1) == hv[i]
for j in rh:
    for k in r9:
        m += lpSum(v[i][j][k] for i in rw) <= 1
    m += addsum(0, j, 1, 0) == hh[j]
%time m.solve()
for j in rh:
    for i in rw:
        print('*' if ch[j][i] == '*' else '%d' % int(value(r[i][j]) + 0.5), end=' ')
    print()
```

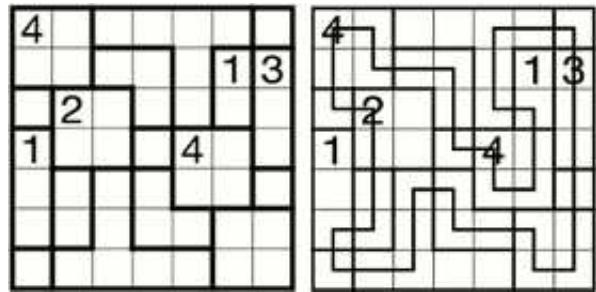
Wall time: 116 ms

```
1 6 7 8 * 9
7 * 8 3 5 4
4 5 9 * 6 *
* 9 * 7 1 2
9 1 3 4 * 5
2 * 5 6 9 8
```

# カントリーロードの解き方

データ (data/countryroad.txt)

## 問題



- いくつかのマスに線を引き、1つの輪を作ります
- 線は、交差や枝分かれしてはいけません
- 太線で区切られたところ(国と呼ぶ)すべてを1回ずつだけ通ります
- 数字は、その数字がある国を線が通るマス数を表します
- 数字のない国には、何マスでもよいです
- 線が通らないマスが、太線(国境)をはさんでタテヨコに隣接してはいけません

## 変数

- vh:0:L, 1:R (1)
- vv:0:U, 1:D (2)
- vhs (3)
- vvs (4)
- vr:表示用 (5)

## 制約

- vh,vv,vhs,vvsの関係 (6)
- 入る数と出る数と同じであること (7)
- 1つの輪 (8)
- その他は略 (9)

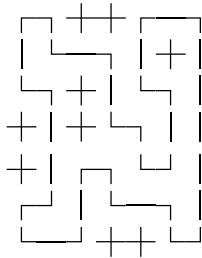
```
In [30]: from collections import defaultdict
from unionfind import *
with open('data/countryroad.txt') as fp:
    nw, nh = [int(s) for s in fp.readline().split(',')]
    rw, rh, rw1, rh1, r2, r4 = range(nw), range(nh), range(nw + 1), range(nh + 1),
    range(2), range(4)
    ch = [fp.readline() for j in rh]
    hh = fp.readlines()
m = LpProblem()
vh = [[[addbinvar() for k in r2] for j in rh] for i in rw1] # 0:L, 1:R (1)
vv = [[[addbinvar() for k in r2] for j in rh1] for i in rw] # 0:U, 1:D (2)
vhs = [[addvar() for j in rh] for i in rw1] # (3)
vvs = [[addvar() for j in rh1] for i in rw] # (4)
vr = [[addvar() for j in rh] for i in rw] # (5)
dic1, dic2 = defaultdict(list), defaultdict(list)
for i in rw:
    m += vvs[i][0] == 0 # (9)
    m += vvs[i][nh] == 0 # (9)
for j in rh1:
    for i in rw:
        m += lpSum(vv[i][j]) == vvs[i][j] # (6)
        m += vvs[i][j] <= 1 # (9)
    if j == nh: break
```

```

m += vhs[0][j] == 0 # (9)
m += vhs[nw][j] == 0 # (9)
for i in rw1:
    m += IpSum(vh[i][j]) == vhs[i][j] # (6)
    m += vhs[i][j] <= 1 # (9)
for i in rw:
    m += IpDot([-1, 1, 1, -1], [vh[i + k][j][l] for k in r2 for l in r2]) !=
        IpDot([1, -1, -1, 1], [vv[i][j + k][l] for k in r2 for l in r2]) # (7)
)
I = [vvs[i][j], vhs[i][j], vvs[i][j + 1], vhs[i + 1][j]]
dic1[ch[j][i]].extend(I)
m += IpDot([2**k for k in r4], I) == vr[i][j] # (9)
m += IpSum(I) <= 2
if i < nw - 1 and ch[j][i] != ch[j][i + 1]:
    m += vr[i][j] + vr[i + 1][j] >= 1 # (9)
    dic2[ch[j][i]].append(vh[i + 1][j][0])
    dic2[ch[j][i + 1]].append(vh[i + 1][j][1])
if j < nh - 1 and ch[j][i] != ch[j + 1][i]:
    m += vr[i][j] + vr[i][j + 1] >= 1 # (9)
    dic2[ch[j][i]].append(vv[i][j + 1][0])
    dic2[ch[j + 1][i]].append(vv[i][j + 1][1])
for s in hh:
    ss = s.split(',')
    if len(ss) < 2: break
    m += IpSum(dic1[ss[0]]) == 2 * int(ss[1]) # (9)
for s in dic2.values():
    m += IpSum(s) == 1 # (9)
while True:
    %time m.solve()
    if m.status != 1: break
    u = unionfind(nw * nh)
    p = -1
    for i in rw:
        for j in rh:
            if value(vhs[i + 1][j]) > 0.5:
                p = i + j * nw
                u.unite(i + j * nw, i + 1 + j * nw)
            if value(vvs[i][j + 1]) > 0.5:
                u.unite(i + j * nw, i + j * nw + nw)
    if all([u.find(i + j * nw) == i + j * nw or u.issame(p, i + j * nw) for i in r
w for j in rh]): break
    for gr in u.groups():
        if len(gr) == 1: continue
        s = []
        for g in gr:
            i, j = g % nw, g // nw
            for k in r2:
                if value(vhs[i + k][j]) > 0.5: s.append(vhs[i + k][j])
                if value(vvs[i][j + k]) > 0.5: s.append(vvs[i][j + k])
        m += IpSum(s) <= len(s) - 2 # (8)
for j in rh:
    for i in rw:
        sys.stdout.write(u'+'+12*' '+'4 | '+'78 '+'—1 '+'34+'+' [int(value(vr[i][j]))]+)
print()

```

Wall time: 152 ms



## カナオレの解き方

データ ([data/kanaore.txt](#))

### 問題

- 全マスに1文字ずつ入れ、リストの言葉全部を盤面に作ります
- どの言葉も、1文字目は言葉の前に書かれている数字のマスに入り、2文字目は矢印の方向に1つ進んだマスに入れます
- 3文字目以降は、タテヨコにつながって入ります
- 1つの文字を複数の言葉が共通して使うこともあります
- 1つの文字を1つの言葉が複数回使うことはありません

			5	
1			4	
				6
2	3			

1①カナオレ 4①ケイスケ  
2①サムライン 5①マツクロ  
3⊖ナンスケ 6①フィルオミノ

ナ	オ	レ	マ	口
カ	ン	ケ	ツ	ク
ラ	イ	イ	ミ	ノ
ム	ケ	ス	オ	フ
サ	ナ	ン	ル	イ

### 変数

- v: 各マスの文字のID (1)
- 各単語ごとに候補(lst)を作り
  - vt: どの候補を選ぶか (2)

### 制約

- 候補から選ぶこと (3)
- 候補を選んだら、その場所は、その文字とすること (4)

```
In [31]: import codecs
with codecs.open('data/kanaore.txt', 'r', 'utf-8') as fp:
    nw, nh, nn = [int(s) for s in fp.readline().strip(u'\ufeff').split(',')]
    rw, rh, rn, r4 = range(nw), range(nh), range(nn), range(4)
    wds = [fp.readline().strip().split(',') for i in rn]
def makecand(lst, x, y, d, l, p, u):
    xx, yy = x + [-1, 0, 1, 0][d], y + [0, -1, 0, 1][d]
    if 0 <= xx < nw and 0 <= yy < nh and (xx, yy) not in u:
        if p == l - 1:
            lst.append(u + [(xx, yy)])
            return
        for k in r4: makecand(lst, xx, yy, k, l, p + 1, u + [(xx, yy)])
m = LpProblem()
v = [[addvar() for j in rh] for i in rw] # (1)
dic, dic2 = {}, {}
for wd in wds:
    for c in wd[3]:
        if not c in dic:
            t = len(dic)
            dic[c], dic2[t] = t, c
    lst = []
    x, y = int(wd[0]), int(wd[1])
    makecand(lst, x, y, u'←↑→↓'.index(wd[2]), len(wd[3]), 1, [(x, y)])
    vt = [addbinvar() for cand in lst] # (2)
    m += lpSum(vt) == 1 # (3)
    for i, cand in enumerate(lst):
        for j, (x, y) in enumerate(cand):
            c = dic[wd[3][j]]
            m += v[x][y] <= c + nw * nh * (1 - vt[i]) # (4)
            m += v[x][y] >= c - nw * nh * (1 - vt[i]) # (4)
%time m.solve()
for j in rh:
    for i in rw:
        print(dic2[int(value(v[i][j]))], end=' ')
print()
```

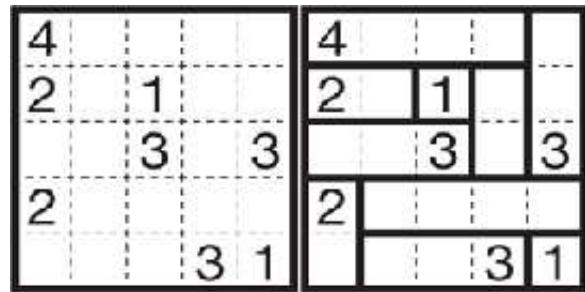
Wall time: 87 ms

ナオレマロ  
カンケツク  
ライイミノ  
ムケスオフ  
サンルイ

# ファイルマットの解き方

データ (data/fillmat.txt)

## 問題



- 点線の上にタテヨコに線を引いて、盤面をいくつかの畳(幅1マスで長さ1~4マスの四角形)に区切れます
- 数字は、その数字を含む畳の面積を、1マスを1として表します
- 数字の入らない畳を作ってもよいが、2つ以上の数字を含む畳を作ってはいけません
- 同じ面積の畳をタテヨコに隣り合わせてはいけません
- 畳の境界線が十字に交差してはいけません

## 変数

- 数字ごとの候補 (1)

## 制約

- 1つの候補を選ぶ (2)
- 4つの角を接する組合せの合計が $\leq 3$  (3)
- 同じ面積の候補同士で隣り合うものの合計 $\leq 1$  (4)

```
In [32]: with open('data/fillmat.txt') as fp:  
    nw, nh = [int(s) for s in fp.readline().split(',')]  
    rw, rh, r4, r19 = range(nw), range(nh), range(4), range(1, 9)  
    ch = [fp.readline() for j in rh]  
    m = LpProblem()  
    vls = [] # list of (var, pos_list)  
    cs = [[LpAffineExpression() for j in rh] for i in rw] # cons of pos  
    dic = {} # key:(x_start, y_start, x_len, y_len), value:(var, pos_list)  
    def chk(v, ky):  
        if ky in dic: m.add(v + dic[ky][0] <= 1) # (4)  
    def cand(i, j, n, dx, dy):  
        p, q = [], []  
        for k in range(n):  
            x, y = i + k * dx, j + k * dy  
            p.append((x, y))  
            if ch[y][x].isdigit(): q.append(int(ch[y][x]))  
        if len(q) >= 2 or (len(q) == 1 and q[0] != n): return  
        v = addbinvar() # (1)  
        for k in range(max(1, n * dy)):  
            chk(v, (i - 1, j + k - n + 1, 0, n))  
            chk(v, (i - n, j + k, n, 0))  
        for k in range(max(1, n * dx)):  
            chk(v, (i + k, j - n, 0, n))  
            chk(v, (i + k - n + 1, j - 1, n, 0))  
        vls.append((v, p))  
        dic[(i, j, dx * n, dy * n)] = vls[-1]  
    for i in rw:  
        for j in rh:  
            for k in r4:  
                if i + k < nw: cand(i, j, k + 1, 1, 0)  
                if k > 0 and j + k < nh: cand(i, j, k + 1, 0, 1)  
    for i, v1 in enumerate(vls):
```

```

for x, y in v1[1]: cs[x][y] += v1[0]
def chk2(ky1, ky2, ky3, ky4):
    if ky1 in dic and ky2 in dic and ky3 in dic and ky4 in dic:
        m.add(dic[ky1][0] + dic[ky2][0] + dic[ky3][0] + dic[ky4][0] <= 3) # (3)
for i in rw:
    for j in rh:
        m += cs[i][j] == 1 # (2)
        if i == 0 or j == 0: continue
        for k1 in r19:
            x1, y1, z1, w1 = (i - k1, j - 1, k1, 0) if k1 < 5 else (i - 1, j - k1
+ 4, 0, k1 - 4)
            if x1 < 0 or y1 < 0: continue
            for k2 in r19:
                x2, y2, z2, w2 = (i, j - 1, k2, 0) if k2 < 5 else (i - 1, j - k2 +
4, 0, k2 - 4)
                if x2 + z2 > nw or y2 < 0: continue
                for k3 in r19:
                    x3, y3, z3, w3 = (i, j, k3, 0) if k3 < 5 else (i, j, 0, k3 - 4
)
                    if x3 + z3 > nw or y3 + w3 > nh: continue
                    for k4 in r19:
                        x4, y4, z4, w4 = (i - k4, j, k4, 0) if k4 < 5 else (i - 1,
j, 0, k4 - 4)
                        if x4 < 0 or y4 + w4 > nh: continue
                        chk2((x1, y1, z1, w1), (x2, y2, z2, w2), (x3, y3, z3, w3),
(x4, y4, z4, w4))
%time m.solve()
i, ss = 65, [[0 for i in rw] for j in rh]
for v1 in vls:
    if value(v1[0]) > 0.5:
        for x, y in v1[1]: ss[y][x] = chr(i)
        i += 1
for s in ss:
    for c in s: print(c, end=' ')
print()

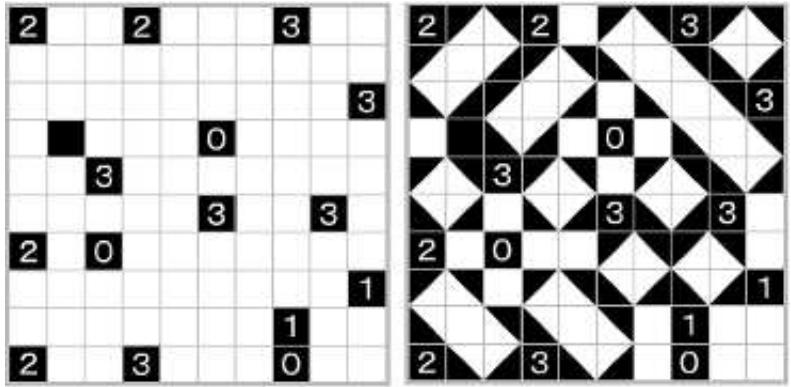
```

Wall time: 83 ms

A A A A I  
B B G H I  
C C C H I  
D E E E E  
D F F F J

# シャカシャカの解き方

データ ([data/shakashaka.txt](#))



## 問題

- 盤面のいくつかの白マスを三角形に黒くぬりつぶします
- マスのぬり方は4通りのいずれかです
- 盤面の数字は、その数字の入っているマスにタテヨコに隣り合うマスのうち、三角形にぬるマスの数を表します
- ぬられずに残った部分は、すべて長方形となります

## 変数

- $va$  (1)

## 制約

- $va_{ij}$  は1つのみ (2)
- 数字か■なら  $va_{ij} \leq 1$  (3)
- 数字なら周りの斜めの合計と同じ (4)
- 画面端で可能な形の指定 (5)
- 右や下との境で合わない物通しの禁止 (6)

```
In [33]: with open('data/shakashaka.txt') as fp:
    nw, nh = [int(s) for s in fp.readline().split(',')]
    rw, rh, r2, r6 = range(nw), range(nh), range(2), range(6)
    ch = [fp.readline() for i in rh]
    m = LpProblem()
    va = [[[addbinvar() for k in r6] for j in rh] for i in rw] # (1)
    def dirs(i, j):
        return [va[i + x - y][j + x + y - 1] for x in r2 for y in r2] #
            if 0 <= i + x - y < nw and 0 <= j + x + y - 1 < nh]
    for i in rw:
        for j in rh:
            v = va[i][j]
            m += lpSum(v) == 1 # (2)
            if ch[j][i] != '.':
                m += v[5] == 1 # (3)
                if ch[j][i].isdigit():
                    m += lpSum(v[1:5] for v in dirs(i, j)) == int(ch[j][i]) # (4)
                if i == 0: m += v[0] + lpSum(v[2:4]) == 0 # (5)
                if j == 0: m += v[0] + lpSum(v[3:5]) == 0 # (5)
                if i == nw - 1: m += v[0] + lpSum(v[1:5:3]) == 0 # (5)
                if j == nw - 1: m += v[0] + lpSum(v[1:3]) == 0 # (5)
                if i > 0:
                    m += lpSum([va[i - 1][j][0]] + va[i - 1][j][1:5:3] + [v[1]] + v[4:6]) <
= 1 # (6)
                    m += lpSum(va[i - 1][j][2:4] + [va[i - 1][j][5]] + [v[0]] + v[2:4]) <=
1 # (6)
                if j > 0:
                    m += lpSum(va[i][j - 1][:3] + v[1:3] + [v[5]]) <= 1 # (6)
                    m += lpSum(va[i][j - 1][3:6] + [v[0]] + v[3:5]) <= 1 # (6)
    %time m.solve()
    for j in rh:
        for i in rw:
            if ch[j][i] != '.': sys.stdout.write(' %c' % ch[j][i])
            else: sys.stdout.write(u' ↗ ↘ ↙ ↖ █'[int(value(lpDot(r6, va[i][j])))])
    print()
```

Wall time: 108 ms

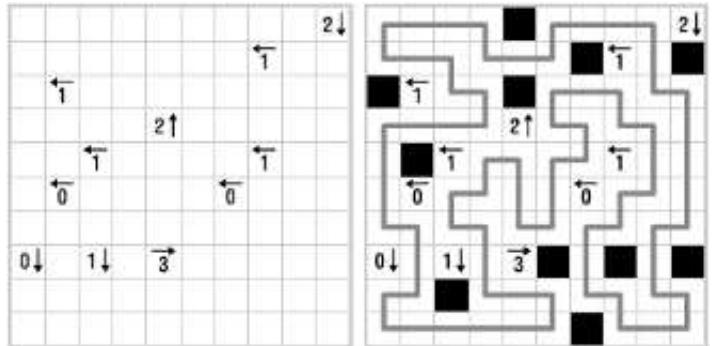
2	2	3	
█	█	█	█
█	█	█	3
█	*	0	█
█	3	█	█
█	█	3	3
2	0	█	█
█	█	█	1
2	3	0	█

# ヤジリンの解き方

データ (data/yajirin.txt)

## 問題

- 線を引いて全体で1つの輪を作り、線が通らないマスは黒くねります
- 数字は、矢印の方向に入る黒マスの数を表します
- 数字のマスに線を引いたり、黒マスにしたりしてはいけません
- 線は、マスの中央を通るようにタテヨコに引き、線を交差させたり、枝分かれさせたりしてはいけません
- 黒マスをタテヨコに連続させてはいけません



## 変数

- 略

## 制約

- 略

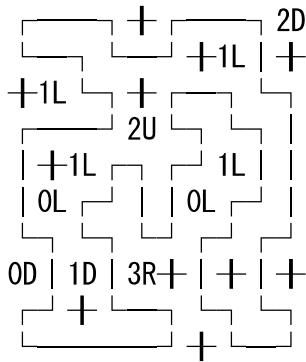
```
In [37]: with open('data/yajirin.txt') as fp:  
    nw, nh = [int(s) for s in fp.readline().split(',')]  
    M, rw, rh, rw1, rh1, r2, r4 = nw * nh, range(nw), range(nh), range(nw + 1), range(nh + 1), range(2), range(4)  
    hh = [s.strip().split(',') for s in fp.readlines()]  
    m = LpProblem()  
    vh = [[[addbinvar() for k in r2] for j in rh] for i in rw1] # 0:L, 1:R  
    vv = [[[addbinvar() for k in r2] for j in rh1] for i in rw] # 0:U, 1:D  
    vhs = [[addvar() for j in rh] for i in rw1]  
    vvs = [[addvar() for j in rh1] for i in rw]  
    vr = [[addvar() for j in rh] for i in rw]  
    vb = [[addbinvar() for j in rh] for i in rw]  
    vd = [[addvar() for j in rh] for i in rw]  
    dic = {}  
    for h in hh:  
        x, y, n, d = int(h[0]), int(h[1]), int(h[2]), 'LURD'.index(h[3])  
        dx, dy = [-1, 0, 1, 0][d], [0, -1, 0, 1][d]  
        m += vr[x][y] == 0  
        m += vb[x][y] == 0  
        m += lpSum(vb[x + dx * i][y + dy * i] for i in range(1, max(nw, nh))) !=  
             if 0 <= x + dx * i < nw and 0 <= y + dy * i < nh) == n  
        dic[(x, y)] = '%d%c' % (n, h[3])  
        if n == 0: px, py = x + dx, y + dy  
    for i in rw:  
        m += vvs[i][0] == 0  
        m += vvs[i][nh] == 0  
    for j in rh1:  
        for i in rw:  
            m += lpSum(vv[i][j]) == vvs[i][j]
```

```

m += vvs[i][j] <= 1
if j == nh: break
m += vhs[0][j] == 0
m += vhs[nw][j] == 0
for i in rw1:
    m += lpSum(vh[i][j]) == vhs[i][j]
    m += vhs[i][j] <= 1
for i in rw:
    m += lpDot([-1, 1, 1, -1], [vh[i + k][j][l] for k in r2 for l in r2]) ==
        lpDot([1, -1, -1, 1], [vv[i][j + k][l] for k in r2 for l in r2])
    l = [vvs[i][j], vhs[i][j], vvs[i][j + 1], vhs[i + 1][j]]
    m += lpDot([2**k for k in r4], l) == vr[i][j]
    m += lpSum(l) <= 2
    if (i, j) not in dic:
        m += vr[i][j] + vb[i][j] >= 1
        m += vr[i][j] <= 12 * (1 - vb[i][j])
    m += vd[i][j] <= M
    if i > 0:
        m += vb[i - 1][j] + vb[i][j] <= 1
        if (i - 1, j) != (px, py):
            m += vd[i - 1][j] + M * (1 - vh[i][j][0]) >= vd[i][j] + 1
        if (i, j) != (px, py):
            m += vd[i][j] + M * (1 - vh[i][j][1]) >= vd[i - 1][j] + 1
    if j > 0:
        m += vb[i][j - 1] + vb[i][j] <= 1
        if (i, j - 1) != (px, py):
            m += vd[i][j - 1] + M * (1 - vv[i][j][0]) >= vd[i][j] + 1
        if (i, j) != (px, py):
            m += vd[i][j] + M * (1 - vv[i][j][1]) >= vd[i][j - 1] + 1
%time m.solve()
for j in rh:
    for i in rw:
        if (i, j) in dic: sys.stdout.write(dic[(i, j)])
        elif value(vb[i][j]) > 0.5: sys.stdout.write(u'+')
        else: sys.stdout.write(u' 12└ 4 ┌ 78 ┌ 1 ┌ 34 ┌' [int(value(vr[i][j]))])
print()

```

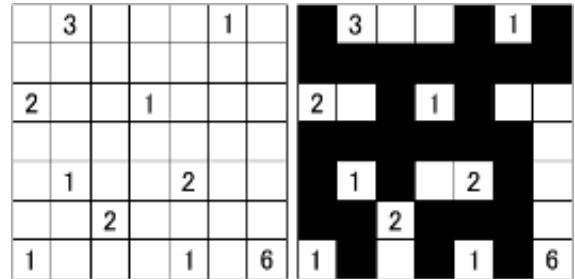
Wall time: 3.06 s



# ぬりかべの解き方

データ ([data/nurikabe.txt](#))

## 問題



- 盤面のいくつかのマスを黒くぬります
- 数字は、黒マスによって分断されたシマのマスの数を表します
- すべてのシマに数字がちょうど1つずつ入ります
- 数字が入っているマスを黒くぬってはいけません
- すべての黒マスはタテヨコにひとつながりになります
- 黒マスを $2 \times 2$ 以上のカタマリにしてはいけません

## 考え方

- 高速に解くために黒マスの数を最大化します

## 変数

- vb:0:white, 1:black (1)
- 島ごとに可能な候補を作ります
  - vt (2)

## 制約

- 黒マスを $2 \times 2$ 以上のカタマリ禁止 (3)
- 候補から1つ選びます (4)
- 候補を選んだらそのマスは白マスで周りは黒マス (5)
- 黒マスが連結していること (6)

```
In [34]: from unionfind import *
with open('data/nurikabe.txt') as fp:
    nw, nh = [int(s) for s in fp.readline().split(',')]
    rw, rh, r2 = range(nw), range(nh), range(2)
    ch = [fp.readline() for j in rh]
m = LpProblem('', LpMaximize)
vb = [[addbinvar() for j in rh] for i in rw] # 0:white, 1:black (1)
m += lpSum(v for vv in vb for v in vv) # onj func
def dirs(i, j):
    return [(i + x - y, j + x + y - 1) for x in r2 for y in r2 if
            0 <= i + x - y < nw and 0 <= j + x + y - 1 < nh]
def make(lst, i, j, n, w):
    if len(w) == n:
        lst.append(w)
        return
    for a, b in dirs(i, j):
        if (a, b) not in w and all([(c, d) == w[0] or ch[d][c] == '.' for c, d in
                                     dirs(a, b)]):
            make(lst, a, b, n, w + [(a, b)])
for i in rw:
    for j in rh:
        if i < nw - 1 and j < nh - 1:
            m += lpSum(vb[i + k][j + l] for k in r2 for l in r2) <= 3 # (3)
        if ch[j][i] == '.': continue
        lst = []
        make(lst, i, j, int(ch[j][i]), [(i, j)])
        lst = [u[0] for u in itertools.groupby(lst)]
        vt = [addbinvar() for w in lst] # (2)
        m += lpSum(vt) == 1 # (4)
        for k, w in enumerate(lst):
            bd = [(c, d) for a, b in w for c, d in dirs(a, b) if (c, d) not in w]
            bd = list(set(bd))
            m += lpSum(vb[x][y] for x, y in w) + len(bd) - lpSum(vb[x][y] for x, y
                         in bd) <= len(w) + len(bd)) * (1 - vt[k]) # (5)
while True:
    %time m.solve()
    if unionfind.isconnected([[value(vb[i][j]) > 0.5 for j in rh] for i in rw]):
        break
    m += lpSum(vb[i][j] for i in rw for j in rh if value(vb[i][j]) < 0.5) >= 1 # (6)
for j in rh:
    for i in rw: print(ch[j][i] if ch[j][i] != '.' else '.' if value(vb[i][j]) < 0.5 else '#', end=' ')
    print()
```

Wall time: 62 ms

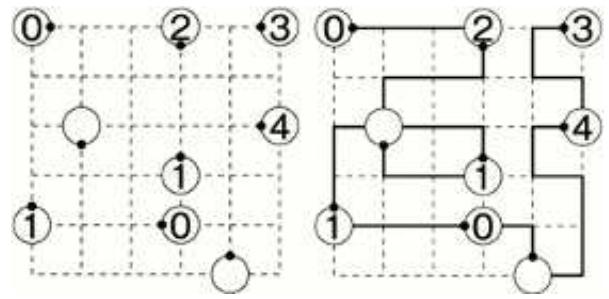
```
# 3 . . # 1 #
# # # # # #
2 . # 1 # . .
# # # # # .
# 1 # . 2 # .
# # 2 # # #
1 # . # 1 # 6
```

# ホタルビームの解き方

データ (data/hotaru.txt)

## 問題

- 全ての白丸の黒点から点線上に線をのばして白丸の黒点でないところにつなげます
- どの線も白丸以外のところで、途切れたり交差したり枝分かれしたりしてはいけません
- 線で全体がひとつつながりにならなければいけません
- 数字は、その白丸の黒点から出る線が白丸につながるまでに曲がる回数を表します



## 変数

- 省略

## 制約

- 省略

```
In [39]: from unionfind import *
L = 9 # limit length from '?'
with open('data/hotaru.txt') as fp:
    nw, nh, nn = [int(s) for s in fp.readline().split(',')]
    rw, rh, rw1, rh1, rn, r4 = range(nw), range(nh), range(nw - 1), range(nh - 1),
    range(nn), range(4)
    hh = [fp.readline().strip().split(',') for k in rn] # x, y, turn, dir
    bh = [[(-1, 0) for j in rh] for i in rw] # (id, dir) of hint
    for k, h in enumerate(hh):
        hh[k] = hc = [int(h[0]), int(h[1]), -2 if h[2] == '?' else int(h[2]), 'LTRB'.index(h[3])]
        bh[hc[0]][hc[1]] = (k, hc[3])
    m = LpProblem()
    cc = [[[LpAffineExpression() for j in rh] for i in rw], # check node overlap
           [[LpAffineExpression() for j in rh] for i in rw1], # check h_line overlap
           [[LpAffineExpression() for j in rh1] for i in rw]] # check v_line overlap
    vas = []
    def make(cands, x, y, n, d, p0, q0):
        dx, dy = [-1, 0, 1, 0][d], [0, -1, 0, 1][d]
        nx, ny = x + dx, y + dy
        if n == -1 or (nx, ny) in p0 or not (0 <= nx < nw and 0 <= ny < nh) or len(p0) > L: return
        p, q = p0 + [(nx, ny)], q0 + [(0, nx, ny), (1, nx, ny), (0, x, y), (1, x, y)][d : d + 1]
        if bh[nx][ny][0] >= 0:
            if n <= 0 and d != (bh[nx][ny][1] + 2) % 4:
                cands.append((p, q))
        return
    for k in r4: make(cands, nx, ny, n if d == k else n - 1, k, p, q)
for h in hh:
    cands = []
    make(cands, h[0], h[1], h[2], h[3], [(h[0], h[1])], [])
    vv = [addbinvar() for cand in cands]
```

```

m += IpSum(vv) == 1
for i in range(len(cands)):
    vas.append((vv[i], cands[i]))
    for j, (w, x, y) in enumerate(cands[i][1]):
        cc[0][cands[i][0][j][0]][cands[i][0][j][1]] += vv[i]
        cc[w + 1][x][y] += vv[i]
for i in rw:
    for j in rh:
        m += cc[0][i][j] <= 1
        if i < nw - 1: m += cc[1][i][j] <= 1
        if j < nh - 1: m += cc[2][i][j] <= 1
while True:
    %time m.solve()
u = unionfind(nn)
l = []
for va, cand in vas:
    if value(va) > 0.5:
        l.append(va)
        (x1, y1), (x2, y2) = cand[0][0], cand[0][-1]
        u.unite(bh[x1][y1][0], bh[x2][y2][0])
if all([u.issame(0, k) for k in rn]): break
m += IpSum(l) <= len(l) - 1
bd = [[' '] * (2 * nw - 1) for j in range(2 * nh - 1)]
for h in hh: bd[h[1] * 2][h[0] * 2] = '?' if h[2] < 0 else str(h[2])
for va, cand in vas:
    if value(va) > 0.5:
        for w, x, y in cand[1]: bd[2 * y + w][2 * x + 1 - w] = '-|'[w]
for b in bd: print(''.join(b))

```

Wall time: 49 ms

Wall time: 48 ms

```

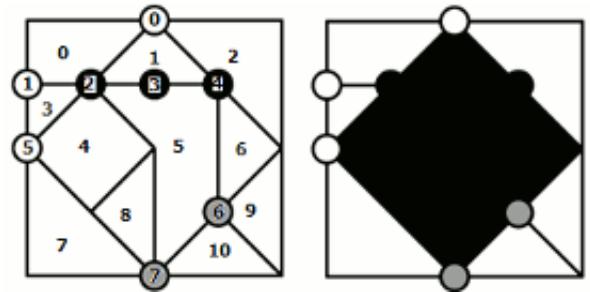
0 - - - 2 - - 3
      |   |
      - - - -
      |       |
      - ? - - - - 4
      |   |   |   |
      - - - 1 - -
      |           |
1 - - - 0 - - - -
      |   |
      ? - -

```

# ステンドグラスの解き方

データ (data/stainedglass.txt)

## 問題



- 線で区切られた部分(ピース)のいくつかを黒くぬります
- 小さな丸は、その丸が接している周囲のピースのうち、黒ピースと白ピースのどちらの数が多いかを表します
- 黒丸なら黒ピースの方が多く、白丸なら白ピースの方が多く、灰色の丸は、同数となります

## 変数

- v: パネルごとに黒く塗るかどうか (1)

## 制約

- 白丸、黒丸、灰色丸ごとに設定 (2)

```
In [40]: with open('data/stainedglass.txt') as fp:  
    nn, np = [int(s) for s in fp.readline().split(',')]  
    rn, rp, r2 = range(nn), range(np), range(2)  
    hh = [fp.readline().split(',') for i in rn]  
    m = LpProblem()  
    v = [addbinvar() for j in rp] # (1)  
    for i in rn:  
        l = [v[int(s)] for s in hh[i][1:]]  
        if hh[i][0] == 'W':  
            m += lpSum(l) <= (len(l) - 1) // 2 # (2)  
        elif hh[i][0] == 'B':  
            m += lpSum(l) >= (len(l) + 2) // 2 # (2)  
        else:  
            m += lpSum(l) == len(l) // 2 # (2)  
    %time m.solve()  
    for j in rp:  
        if value(v[j]) > 0.5: print(j, end=' ')  
    print()
```

Wall time: 50 ms

1 4 5 6 8

# さとがえりの解き方

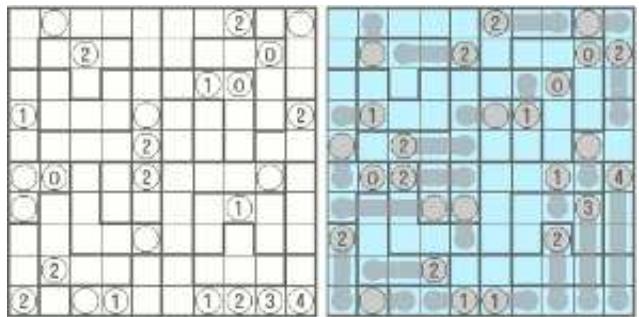
データ ([data/satogaeri.txt](#))

## 問題

- をタテヨコいずれかにまっすぐ移動し、すべての○が、太線で

区切られたところ(国)1つにつき1つずつ入るようにします

- の中の数字は、移動するマス数を表します
- 数字のない○は何マス移動するか不明(移動しないこともあります)
- が移動した跡や、他の○のあるところには○を移動できません



## 変数

- 省略

## 制約

- 省略

```
In [41]: from collections import defaultdict
with open('data/satogaeri.txt') as fp:
    nw, nh, nn = [int(s) for s in fp.readline().split(',')]
    mx, rw, rh, rn, r2 = max(nw, nh) + 1, range(nw), range(nh), range(nn), range(2)
    ch = [[c for c in fp.readline()] for j in rh]
    hh = [list(map(int, fp.readline().split(','))) for k in rn]
dic = defaultdict(list)
for i in rw:
    for j in rh: dic[ch[j][i]].append((i, j))
for x, y, n in hh: ch[y][x] = '*'
def chk(xy): return 0 <= xy[0] < nw and 0 <= xy[1] < nh and ch[xy[1]][xy[0]] != '*'
m = LpProblem()
vls = []
vo = [[[[] for j in rh] for i in rw]
cc = [[LpAffineExpression() for j in rh] for i in rw]
for x, y, n in hh:
    cands = []
    if n == 0: cands.append([(x, y)])
    for p in r2:
        for q in r2:
            dx, dy = p - q, p + q - 1
            l = list(itertools.takewhile(chk, [(x + dx * k, y + dy * k) for k in range(1, 1 + n if n >= 0 else mx)]))
            if len(l) > 0 and (n < 0 or len(l) == n):
                for k in range(0 if n < 0 else len(l) - 1, len(l)):
                    cands.append([(x, y)] + l[:k + 1])
    v = [addbinvar() for cand in cands]
    m += lpSum(v) == 1
    for k, cand in enumerate(cands):
        vls.append((v[k], cand))
        for i, j in cand: cc[i][j] += v[k]

        vo[cand[-1][0]][cand[-1][1]].append(v[k])
for i in rw:
    for j in rh:
        m += cc[i][j] <= 1
for c, lst in dic.items():
    m += lpSum(lpSum(vo[i][j] for i, j in lst)) == 1
%time m.solve()
for vl, cand in vls:
    if value(vl) > 0.5:
        d = 1 if cand[0][0] == cand[-1][0] else 0
        for i, j in cand[:-1]: ch[j][i] = '-|'[d]
        ch[cand[-1][1]][cand[-1][0]] = '*'
for s in ch: print(''.join(s), end=' ')

```

Wall time: 40 ms

```
a|aaa*--*
a*--*bff**
ghehij|*f|
-*hh-**jf|
*i*-klj*m
|**-rr*|*
---**ru|*|
*too|ru*||
|---*xyw|||
|*---**-|||
```

# スケルトンの解き方

データ (data/skeleton.txt)

## 問題

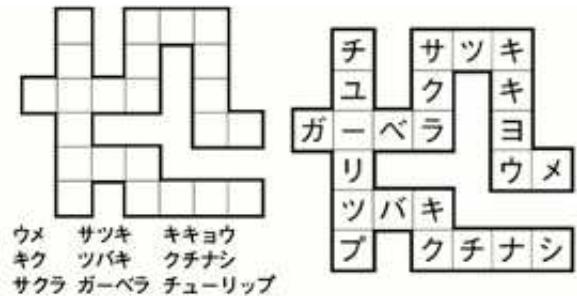
- リストの言葉を全て、盤面にちょうど1つ入れます
- 参考

## 変数

- 省略

## 制約

- 省略



```
In [42]: import codecs
with codecs.open('data/skeleton.txt', 'r', 'utf-8') as fp:
    nw, nh, nn = [int(s) for s in fp.readline().strip(u'\ufeff').split(',')]
    rw, rh, rn, r4 = range(nw), range(nh), range(nn), range(4)
    ch = [fp.readline() for j in rh]
    wds = [fp.readline().strip() for i in rn]
m = LpProblem()
vo = [[addvar() for j in rh] for i in rw]
dic, dic2, dic3 = {}, {}, {}
for wd in wds:
    for c in wd:
        if c not in dic: dic[c], dic2[len(dic2)] = len(dic2), c
    l = dic3.setdefault(len(wd), ([], []))
    l[0].append(wd)
M = len(dic) + 1
def chk(xy): return 0 <= xy[0] < nw and 0 <= xy[1] < nh and ch[xy[1]][xy[0]] != '.'
for i in rw:
    for j in rh:
        if i == 0 or ch[j][i - 1] == '.':
            sp = list(itertools.takewhile(chk, [(i + k, j) for k in rw]))
            if len(sp) > 1: dic3[len(sp)][1].append(sp)
        if j == 0 or ch[j - 1][i] == '.':
            sp = list(itertools.takewhile(chk, [(i, j + k) for k in rw]))
            if len(sp) > 1: dic3[len(sp)][1].append(sp)
for nl, (wl, pl) in dic3.items():
    nc = len(wl)
    vb = [[addbinvar() for j in range(nc)] for i in range(nc)]
    for i in range(nc):
        m += lpSum(vb[i]) == 1
        m += lpSum(vb[k][i] for k in range(nc)) == 1
        for j in range(nc):
            wd = wl[j]
            for k in range(nl):
                (x, y), a = pl[i][k], dic[wd[k]]
                m += vo[x][y] <= a + M * (1 - vb[i][j])
                m += vo[x][y] >= a - M * (1 - vb[i][j])
%time m.solve()
for j in rh:
    for i in rw:
        print('' if ch[j][i] == '.' else dic2[int(value(vo[i][j]))], end=' ')
    print()
```

Wall time: 38 ms  
 チ サツキ  
 ュ ク キ  
 ガ 一 ベラ ヨ  
 リ ラ  
 ツバキ  
 プ クチナシ

# 数理最適化で解ける様々なパズル

カックロ	ののぐらむ	美術館	ナンバーリンク	覆面算
不等式	ビルディングパズル	ウォールロジック	波及効果	ナンバースケルトン
スリザーリンク	四角に切れ	ましゅ	橋をかけろ	のりのり
ブロックパズル	タイルペイント	因子の部屋	黒どこ	推理パズル
ひとりにしてくれ	へやわけ	ペイントエリア	数コロ	パイリンク
クリーク	アイスバーン	サムライン	カントリーロード	カナオレ
フィルマット	シャカシャカ	ヤジリン	ぬりかべ	ホタルビーム
ステンドグラス	さとがえり	スケルトン	数独	

## まとめ

- 数理モデルをPython上で作成し、解くことができます
- Pythonによる数理モデルは簡潔でわかりやすく、様々な問題を記述できます
- 汎用ソルバでも、多くの問題を扱えますが、規模が大きい問題では、専用のソルバを用い、近似解法を用いた方がよいこともあります
- 今日の資料 <http://goo.gl/kAbWrA> (<http://goo.gl/kAbWrA>)