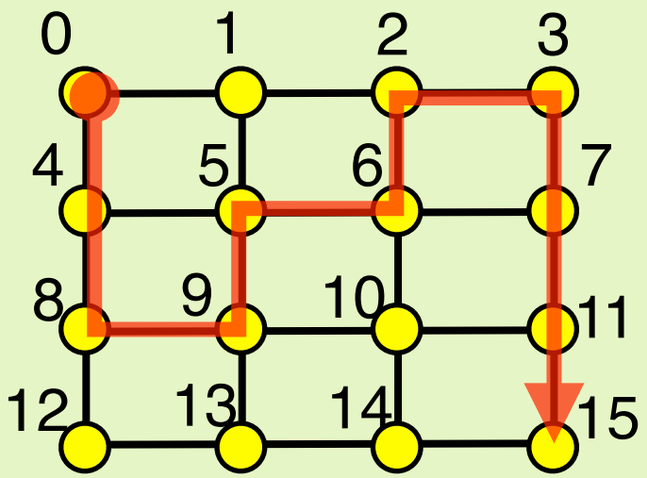
第7回課題 (1)

ｎｘｎ の格子点が下図のように均等に配置されて，上下左右隣同士が線で結合しているとする．このとき，一番左上の点から一番右下の点へ**同じ点を2度通らないで行く行き方**をすべて列挙する関数 kadai(n) を作れ．



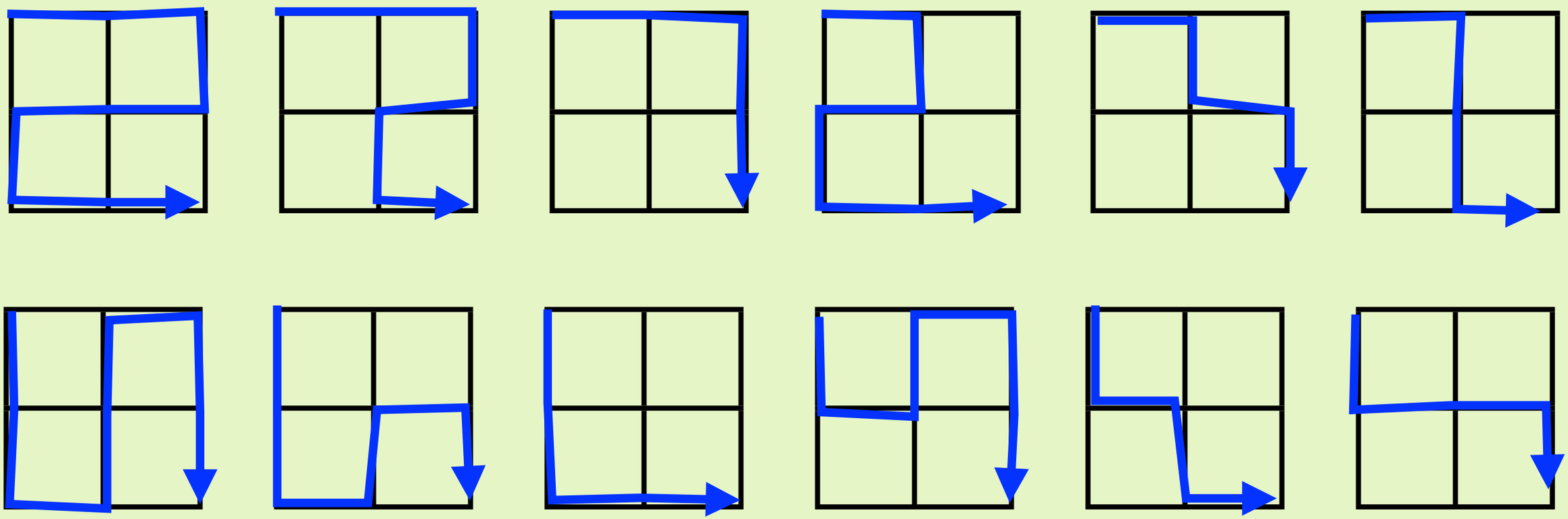

```
[0, 4, 8, 9, 5, 6, 2, 3,
7, 11, 15]
```

ただし，それぞれの格子点は番号が0からn^2 - 1まで付けられているとする．これらの番号の並びが経路となる．ノード k → ノード k − 1 という移動は左へ移動，ノード k → ノード k + 4 は下への移動を表すことになる．ただし，場所によっては移動することができない．

上図の例の場合，赤線の経路は上記のリストによって表現される．関数 kadai(n) が返すべきデータは，これらの経路のリストをさらにリストで集めたものである．すなわち [[0, 4, 8, ...], [0, 1, 5, ...], ...] のような形をしたリストを返せば良い．

第7回課題 (2)

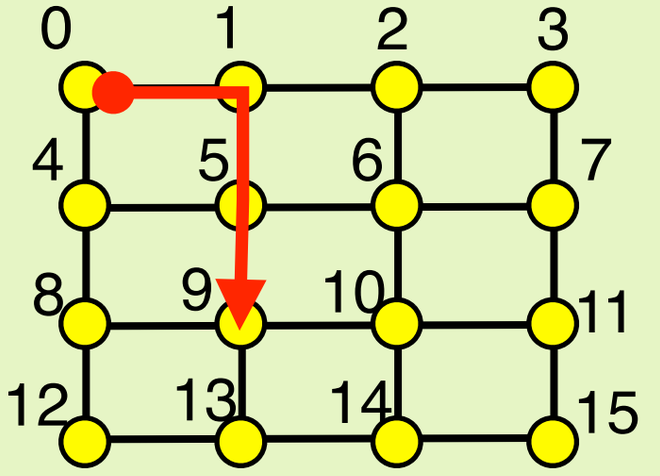
たとえば，n = 3の場合，「田」の字上の経路を示しており，すべての経路は以下の12通りである．結局答えとしては，下のような配列を返せば良い．



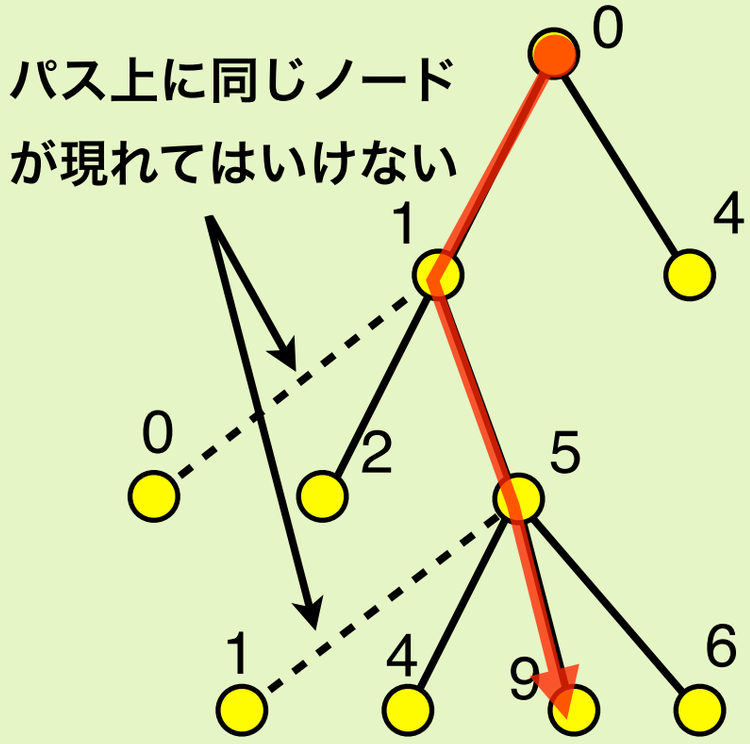
```
[[0,1,2,5,4,3,6,7,8], [0,1,2,5,4,7,8], [0,1,2,5,8],
[0,1,4,3,6,7,8], [0,1,4,5,8], [0,1,4,7,8], [0,3,4,1,2,5,8],
[0,3,4,5,8], [0,3,4,7,8], [0,3,6,7,4,1,2,5,8],
[0,3,6,7,4,5,8], [0,3,6,7,8]]
```

第7回課題 (3)

この問題は深さ優先探索で解くことができる．経路にそって，以下のような木を考える．この木を辿ることによって，15まで辿り着けば一つの解となる．




```
[0, 4, 8, 9, 5, 6, 2, 3,
7, 11, 15]
```



n	解の個数
3	12
4	184
5	8512
6	1262816
7	???
8	???

配列の扱いについて (1)

JavaScriptの配列はCの配列にくらべてかなり柔軟に操作可能なので，その機能を使ってプログラミングするのが良い．授業のプリントにも一部書かれているが，以下に特殊な使い方を幾つかまとめておく．

配列の拡張： 配列の最後に要素を付け加えるには，push()を用いる．

```
js> a = [1, 2, 3, 4]
[1, 2, 3, 4]
js> a.push(6)
5
js> a
[1, 2, 3, 4, 6]
```

最後に相当する位置に代入することによって拡張させることもできる．

```
js> a = [1, 2, 3, 4]
[1, 2, 3, 4]
js> a[a.length] = 6
6
js> a
[1, 2, 3, 4, 6]
```

配列の扱いについて (2)

配列の縮小： 配列の最後の要素を削除するには，pop()を用いる.

```
js> a = [1, 2, 3, 4]
[1, 2, 3, 4]
js> a.pop()
4
js> a
[1, 2, 3]
```

また，強制的に長さを小さくすることで縮小させることもできる.

```
js> a = [1, 2, 3, 4]
[1, 2, 3, 4]
js> a.length = 3
3
js> a
[1, 2, 3]
```

配列の扱いについて (3)

配列のコピー： 新たな配列をつくり，その中身を与えられた配列を同じにするには，`slice(0)`を用いる．

```
js> a = [1, 2, 3, 4]
[1, 2, 3, 4]
js> b = a.slice(0)
[1, 2, 3, 4]
js> a[2] = 23
23
js> a
[1, 2, 23, 4]
js> b
[1, 2, 3, 4]
```

解答例 (1)

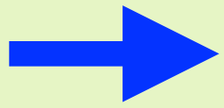
```
function kadai(n){
    var answer = []
    function is_used_in(aa, k){
        for (var i = 0; i < aa.length; i++)
            if (aa[i] == k) return true
        return false
    }
    function call_route(aa, k){
        var m = aa.slice(0)
        m.push(k)
        route(m)
    }
    function route(aa){
        var i = aa[aa.length - 1]
        if (i == n * n - 1) answer.push(aa)
        else {
            if (i >= n && ! is_used_in(aa, i - n)) call_route(aa, i - n)
            if (i % n != 0 && ! is_used_in(aa, i - 1)) call_route(aa, i - 1)
            if ((i + 1) % n != 0 && ! is_used_in(aa, i + 1))
                call_route(aa, i + 1)
            if (i < n * n - n && ! is_used_in(aa, i + n)) call_route(aa, i + n)
        }
    }
    route([0])
    return answer
}
```

深さ優先探索によってすべての解をみつける.

解答例 (2)

実際に動かしてみると以下のようになる.

```
for (i = 2;  i < 7; i++){
    var a = kadai(i).length
    puts(i + " " + a)
}
```



```
2 2
3 12
4 184
5 8512
6 1262816
```

kadai(7)はこの方法で数え上げるのは不可能であると考えられる.