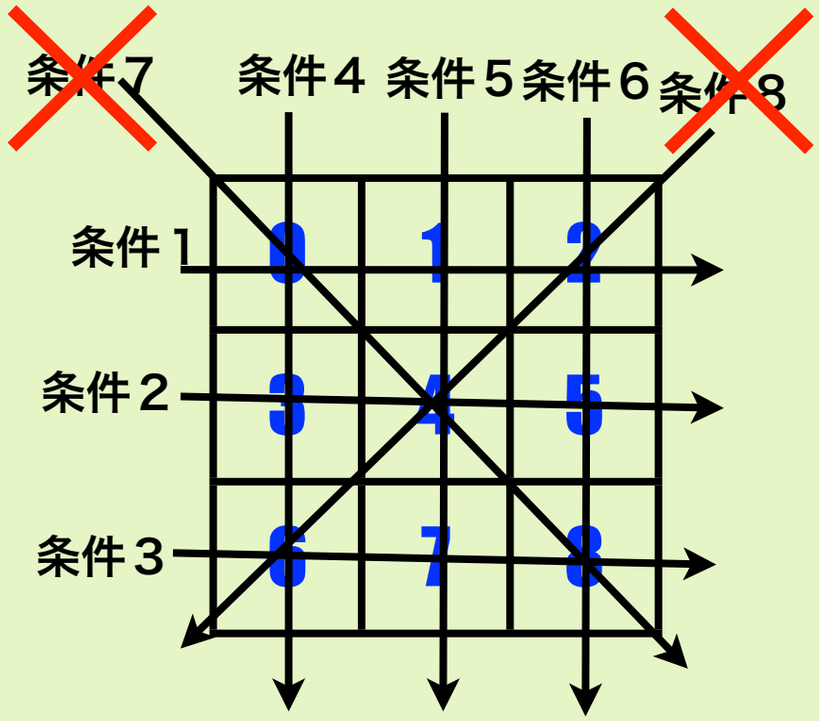
課題 1

一般の**魔方陣 (magic square)** は右図のように縦，横，斜めの和がすべて等しいという条件の成り立つ0から8までの数の並びである．このパズルを変形して，右図の斜めの条件を除外して，縦および横の条件が成り立っている並びのことを**準魔方陣（semi-magic square)**と呼ぶ．4x4の準魔方陣（すなわち，0から15までの数のならび）のうち，左上の数がnであるものの総数を返す関数 kadai1(n) を作れ.



ヒント： なるべく早い段階で枝刈りをするようなプログラムにすると，少ない計算量で数えることができる.

課題1解答例 (1)

```
function kadai(n){
    function check(pat, i, j, k, l){
        var sum = pat[i] + pat[j] + pat[k] + pat[l]
        if (sum != 30) return true
        else return false
    }
    function magic(pat, rest){
        var llen = pat.length
        if (llen >= 16) return 1
        if (llen == 4 && check(pat, 0, 1, 2, 3)) return 0
        if (llen == 8 && check(pat, 4, 5, 6, 7)) return 0
        if (llen == 12 && check(pat, 8, 9, 10, 11)) return 0
        if (llen == 13 && check(pat, 0, 4, 8, 12)) return 0
        if (llen == 14 && check(pat, 1, 5, 9, 13)) return 0
        if (llen == 15 && check(pat, 2, 6, 10, 14)) return 0
        var sum = 0
        for (var i = 0; i < rest.length; i++){
            var ppat = pat.slice(0)
            var restx = rest.slice(0)
            ppat.push(rest[i])
            restx.splice(i, 1)
            sum += magic(ppat, restx)
        }
        return sum
    }
    var rest = []
    for (var i = 0; i < 16; i++)
        if (i != n) rest.push(i)
    return magic([n], rest)
}
```

例題をそのままプログラムにしたもの．時間がかなりかかることになる．

以下のように実行する

```
puts(kadai(0))
```

↓

34344

引数に0から15のどれをいれても同じ値になる

課題1解答例 (2)

```
function kadai(n){
    var pat = new Array(16)
    var used = new Array(16)
    function check(pat, i, j, k, l){
        var sum = pat[i] + pat[j] + pat[k] + pat[l]
        if (sum != 30) return true
        else return false
    }
    function magic(i){
        if (i >= 16) return 1
        if (i == 4 && check(pat, 0, 1, 2, 3)) return 0
        if (i == 8 && check(pat, 4, 5, 6, 7)) return 0
        if (i == 12 && check(pat, 8, 9, 10, 11)) return 0
        if (i == 13 && check(pat, 0, 4, 8, 12)) return 0
        if (i == 14 && check(pat, 1, 5, 9, 13)) return 0
        if (i == 15 && check(pat, 2, 6, 10, 14)) return 0
        var sum = 0
        for (var k = 0; k < 16; k++){
            if (used[k] === false){
                used[k] = true
                pat[i] = k
                sum += magic(i + 1)
                used[k] = false
            }
        }
        return sum
}
```

magicが配列を渡しながら再帰するのは無駄なので，配列は外部で定義しておく．特に現在どの数が使われているのかを配列usedに覚えさせて，適宜その値を変える．magic(i)はpatの0からi - 1番目までが決まって，iつぎに決定するという意味.

```
    for (var i = 0; i < 16; i++){
        used[i] = false
    }
    used[n] = true
    pat[0] = n
    return magic(1)
}
```

課題1解答例 (3)

そもそも束縛が存在することから決めるべき数値は左上の3x3の領域（青い部分）の値である．その部分の値が決まれば，それ以外の場所は自動的にあたいが決まる．xの場所の値が決まると青以外の場所も一部決定される．

0	1	2x	
3	4	5x	
6x	7x	8x	

0の位置は外部から指定されるので，他の8箇所の値を決めればすべて決まることになる．これによって木構造がかなり単純化される．ただし，用いる値は0〜15であり，”x”のついている場所については，2つ以上の数が使用済になる．

課題1解答例 (4)

プログラムは多少複雑になる.

```
function kadai(n){
    var pat = new Array(9)
    var used = new Array(16)
    function idx2(i){
     if (i == 2) return [0, 1]
     else if (i == 5) return [3, 4]
     else if (i == 6) return [0, 3]
     else if (i == 7) return [1, 4]
     }
     function magic(i){
     if (i == 0 || i == 1 || i == 3 || i == 4){
         var sum = 0
         for (var k = 0; k < 16; k++){
         if (used[k] === false){
             used[k] = true
             pat[i] = k
             sum += magic(i + 1)
             used[k] = false
         }
         }
         return sum
     } else if (i == 2 || i == 5 || i == 6 || i == 7 ){
         var [ii1 , ii2] = idx2(i)
         var d = 30 - (pat[ii1] + pat[ii2])
         var sum = 0
         var [st, end] = [d - 15, 15]
         if (d <= 15 && d >= 0)
             [st, end] = [0, d]
         for (var k = st; k <= end; k++){
             var el = d - k
             if ((k != el) && (used[k] === false) &&
              (used[el] === false)){
              used[k] = true;
              used[el] = true;
              pat[i] = k
              sum += magic(i + 1)
              used[k] = false
              used[el] = false
         }
     }
     return sum
}
```

課題1解答例 (5)

ブラウザ上で実行すると1秒かからないで結果がでる.

```
} else if (i == 8){
    var sum = 0
    for (var k = 0; k < 16; k++){
        if (used[k] === true) continue
        used[k] = true
        var dd1 = 30 - (pat[2] + pat[5] + k)
        var dd2 = 30 - (pat[6] + pat[7] + k)
        if (dd1 < 0 || dd2 < 0 || dd1 == dd2 || k == dd1 || k == dd2
            || dd1 > 15 || dd2 > 15){
            used[k] = false
            continue
        }
        if (used[dd1] === true){
            used[k] = false
            continue
        }
        used[dd1] = true
        if (used[dd2] === true){
            used[k] = false
            used[dd1] = false
            continue
        }
                used[dd2] = true
                sum += 1
                used[dd1] = false
                used[dd2] = false
                used[k] = false
            }
            return sum
        }
    }
    for (var i = 0; i  < 16; i++)
        used[i] = false
    used[n] = true
    pat[0] = n
    return magic(1)
}
```

```
var t1 = new Date()
var res = kadai(0)
var t2 = new Date()
puts(res)
puts("Time = " + (t2 - t1) + " ms")
```

```
34344
Time = 314 ms
```

課題2

3組の夫婦がいる．この3組の夫婦は今ボートを使って川の左岸から右岸へ渡りたい．このボートは二人乗りで３人以上は乗れない．さらに複雑な事情として，3組の夫婦のそれぞれの夫は非常に嫉妬深く，自分がいないときに他の夫と自分の妻が一緒に居ることを許さない．このような状況で全員が左岸から右岸に渡るにはどうしたら良いか．ただし，ボートは無人で移動できないとする．幅優先探索を用いて最も回数の少ない渡り方を返す関数 kadai2() を作れ．ただし，返す値は，　夫1, 妻1，夫2, 妻2，夫3, 妻3をそれぞれ0, 1, 2, 3, 4, 5 として，それぞれの時点でボートに乗るメンバーのリストとして，

kadai2() = [[1, 3], [1], [0, 2], ...]

のように返せ．答は複数存在する可能性があるが，そのうちの一つを返せば良い．もちろんボートが無人で移動することはない．

課題２解答例 (1)

```
function kadai2(){
    var init_state = [0, 1, 1, 1, 1, 1, 1]
    var final_state = [1, 0, 0, 0, 0, 0, 0]
    var move_list = [[0], [1], [2], [3], [4], [5],
        [0, 1], [0, 2], [0, 4],
        [1, 3], [1, 5],
        [2, 3], [2, 4],
        [3, 5],
        [4, 5]]
var tabulist = []
function move_state(state, move){
    state1 = state.slice(0)
    var [m1, m2] = [1, 0]
    if (state[0] == 1) [m1, m2] = [0, 1]
    for (var i = 0; i < move.length; i++)
        if (state[move[i] + 1] != m1) return null
    for (var i = 0; i < move.length; i++)
        state1[move[i] + 1] = m2
    state1[0] = m1
    return state1
}
function eq_state(state1, state2){
    for (var i = 0; i < 7; i++)
        if (state1[i] != state2[i]) return false
    return true
}
```

状態は６人が舟がどちらの岸にあるか（左岸：0, 右岸: 1) とそれぞれ人々がどちらの岸にいるか（左岸が1で右岸が0）で表現できる.

夫婦1：夫 (0)，妻 (1)

夫婦2：夫 (2)，妻 (3)

夫婦3：夫 (4)，妻 (5)

課題２解答例 (2)

```
function add_to_tabulist(state){
    for (var i = 0; i < tabulist.length; i++){
        if (eq_state(state, tabulist[i])) return false
    }
    tabulist.push(state)
    return true
}
function check_state(state){
    if (state[1] != state[2]){
        if (state[2] == state[3] || state[2] == state[5]) return false
    }
    if (state[3] != state[4]){
        if (state[4] == state[1] || state[4] == state[5]) return false
    }
    if (state[5] != state[6]){
        if (state[6] == state[1] || state[6] == state[3]) return false
    }
    return true
}
```

課題２解答例 (3)

最後に幅優先探索で問題を解く．11ステップで解くことができる.

```
add_to_tabulist(init_state)
var queue = [[init_state, null, null]]
while (queue.length > 0){
    var node = queue.shift()
    var [state, parent, mv] = node
    if (eq_state(state, final_state)) break
    for (var i = 0; i < move_list.length; i++){
        var state1 = move_state(state, move_list[i])
        if (state1 == null) continue
        if (! check_state(state1)) continue
        if (! add_to_tabulist(state1)) continue
        queue.push([state1, node, move_list[i]])
    }
}
var res = []
while (node != null){
    var [state, parent, mv] = node
    res.unshift(mv)
    node = parent
}
res.shift()
return res
}
```

```
res = kadai2()
for (var i = 0; i < res.length; i++)
  puts(res[i])
```

夫婦1：夫 (0)，妻 (1)

夫婦2：夫 (2)，妻 (3)

夫婦3：夫 (4)，妻 (5)

```
0,1  夫妻１で右岸へ
0    夫１が左岸へ戻る
3,5  妻１と妻３が右岸へ
1    妻１が左岸へ戻る
2,4  夫２と夫４が右岸へ
2,3  夫婦２が左岸へ戻る
0,2  夫１と夫２が右岸へ
5    妻３が左岸へ戻る
1,3  妻１と妻２が右岸へ
1    妻１が左岸へ戻る
1,5  妻１と妻３が右岸へ
```