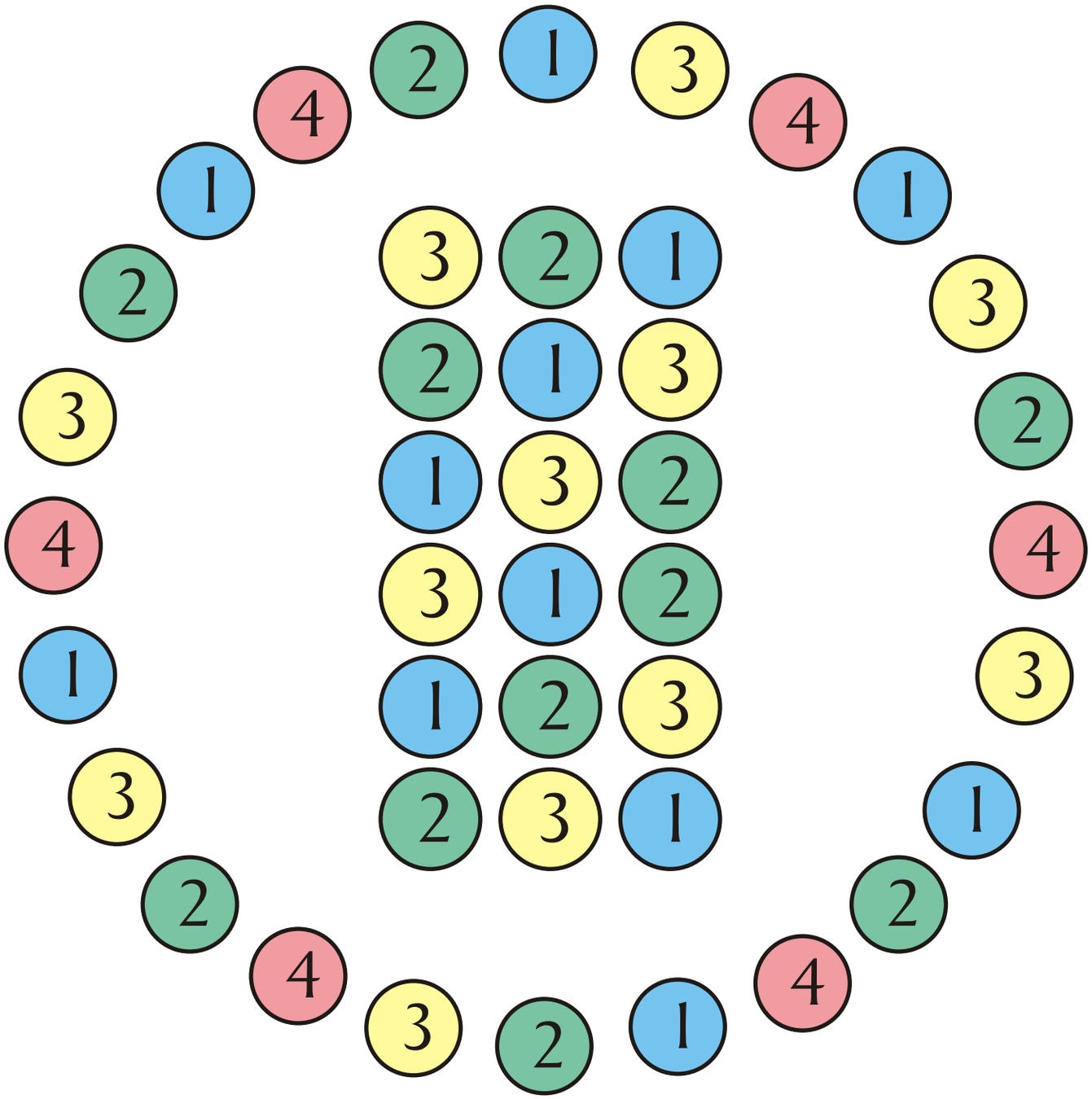
Shorthand Universal Cycles for Permutations

Aaron Williams
University of Victoria

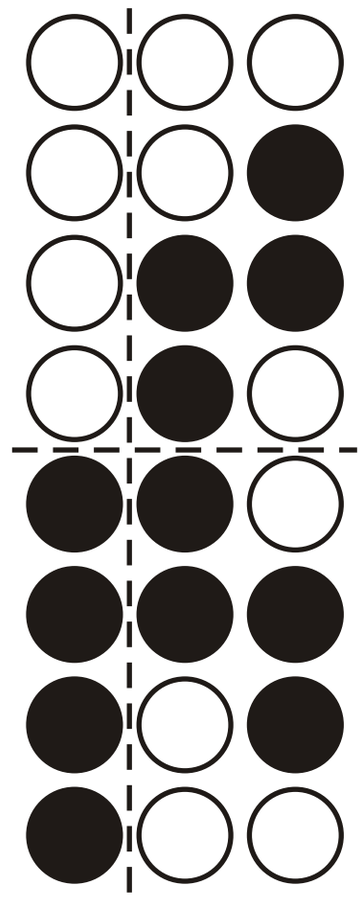


Generating Combinatorial Objects

Generate all permutations, binary strings, trees

Generating Combinatorial Objects

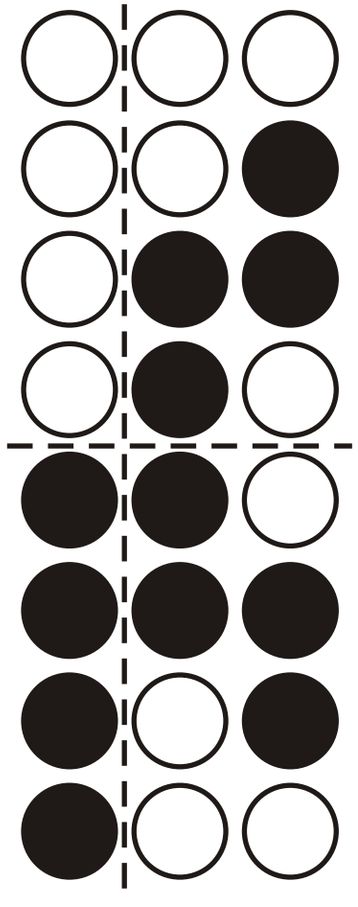
Generate all permutations, binary strings, trees



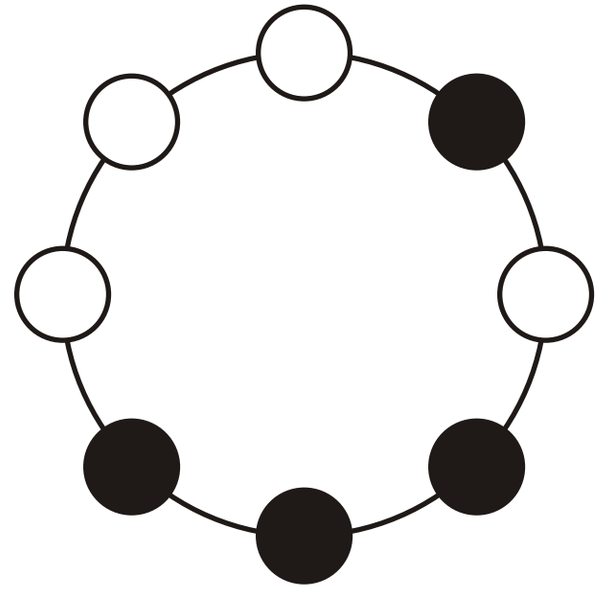
Binary Reflected Gray Code

Generating Combinatorial Objects

Generate all permutations, binary strings, trees



Binary Reflected Gray Code

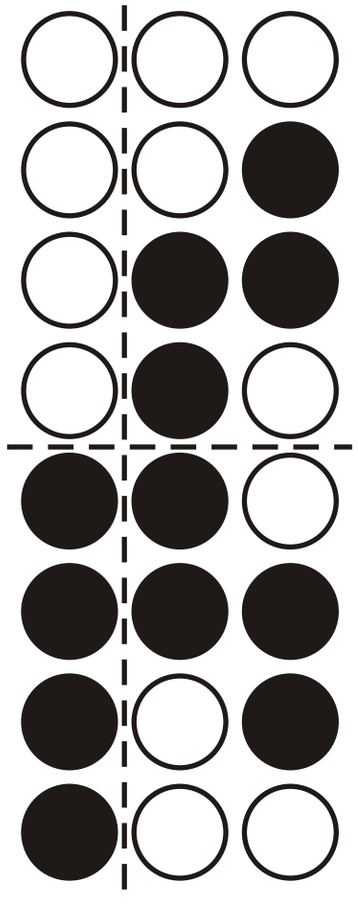


Universal Cycle

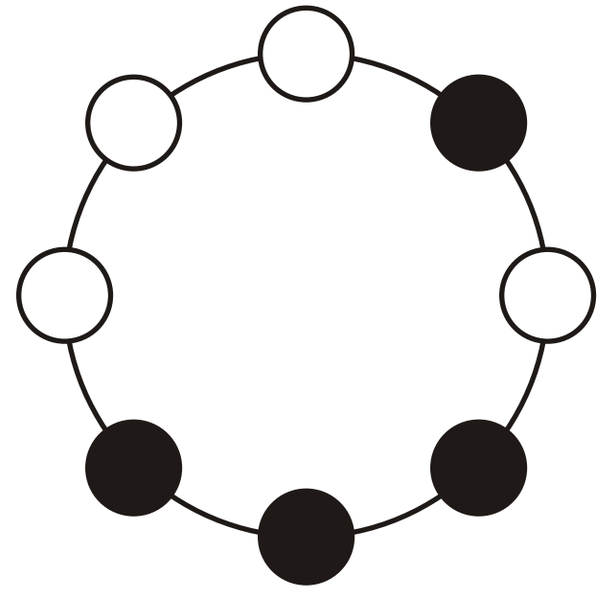
Generating Combinatorial Objects

Generate all permutations, binary strings, trees

Efficiency
- Constant Amortized Time
- Loopless or Loop-free
- Memory



Binary Reflected Gray Code



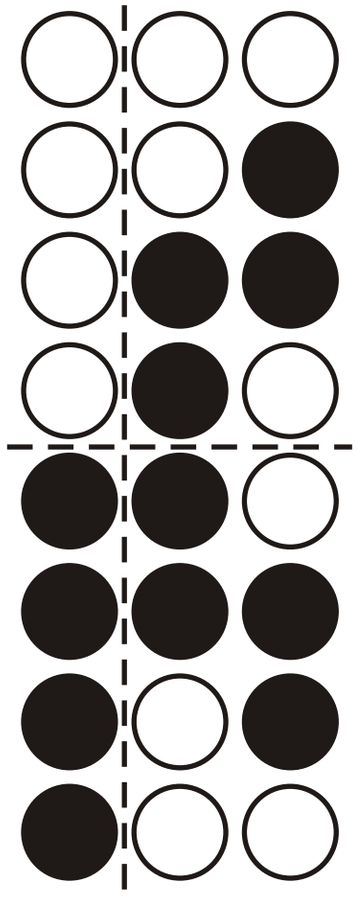
Universal Cycle

Generating Combinatorial Objects

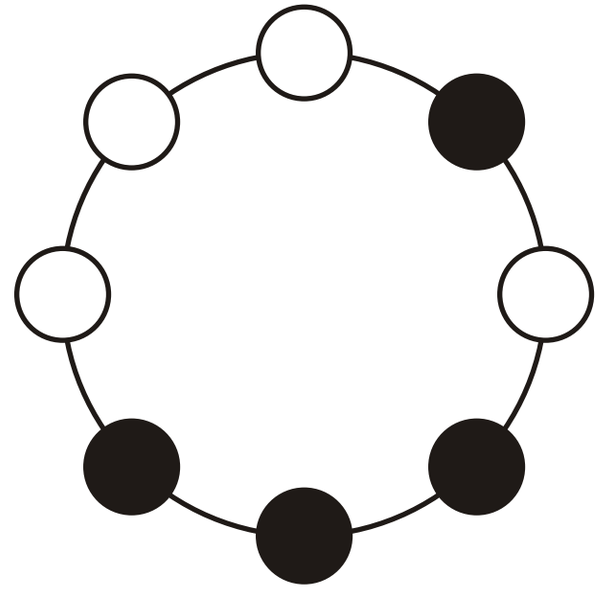
Generate all permutations, binary strings, trees

Efficiency
- Constant Amortized Time
- Loopless or Loop-free
- Memory

Additional Properties
- Predictable changes
- Ranking / unranking
- Induced
- Fairness
- Cyclic
- Mathematical Elegance



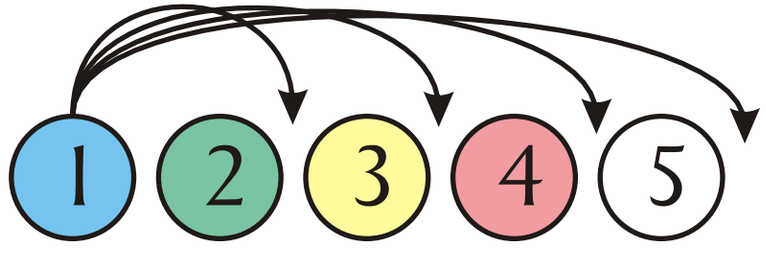
Binary Reflected Gray Code



Universal Cycle

Generating Permutations by Prefix Rotations

Permutations of {1,2,...,n} by prefix rotations Corbett, Jiang-Ruskey, Ruskey-W.



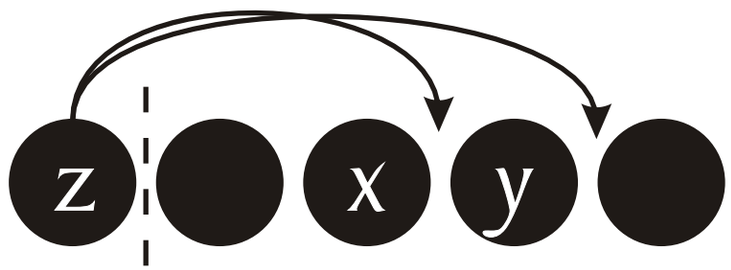
Generating Permutations by Prefix Rotations

Permutations of {1,2,...,n} by prefix rotations Corbett, Jiang-Ruskey, Ruskey-W.

Leftmost increase $x < y$

If $z > x$ then xzy

If $z < x$ then xyz



"Cool-lex"

Generating Permutations by Prefix Rotations

Permutations of {1,2,...,n} by prefix rotations Corbett, Jiang-Ruskey, Ruskey-W.

Leftmost increase $x < y$

If $z > x$ then xzy

If $z < x$ then xyz

example

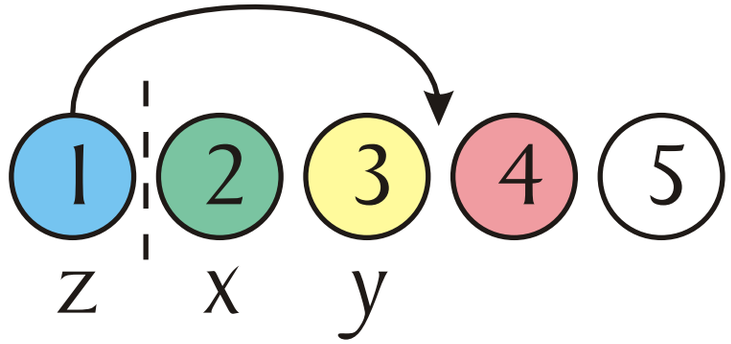
Generating Permutations by Prefix Rotations

Permutations of {1,2,...,n} by prefix rotations Corbett, Jiang-Ruskey, Ruskey-W.

Leftmost increase $x < y$

If $z > x$ then xzy

If $z < x$ then xyz



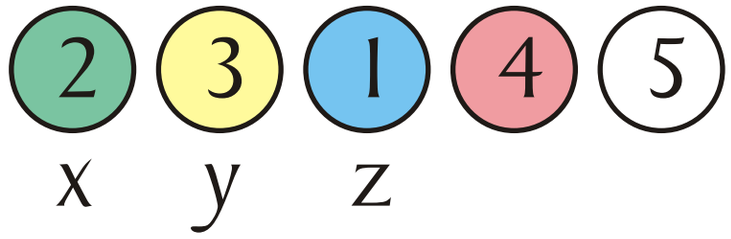
Generating Permutations by Prefix Rotations

Permutations of {1,2,...,n} by prefix rotations Corbett, Jiang-Ruskey, Ruskey-W.

Leftmost increase $x < y$

If $z > x$ then xzy

If $z < x$ then xyz



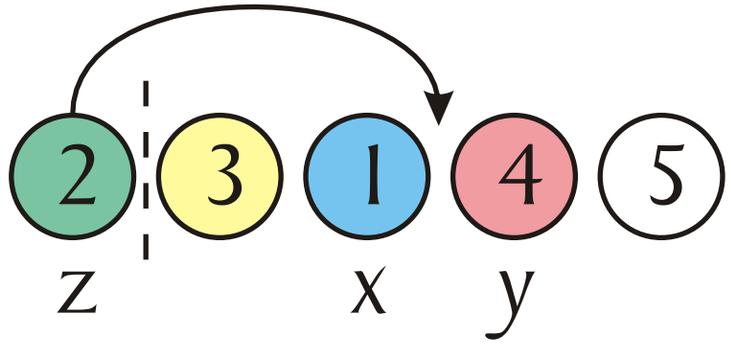
Generating Permutations by Prefix Rotations

Permutations of {1,2,...,n} by prefix rotations Corbett, Jiang-Ruskey, Ruskey-W.

Leftmost increase $x < y$

If $z > x$ then xzy

If $z < x$ then xyz



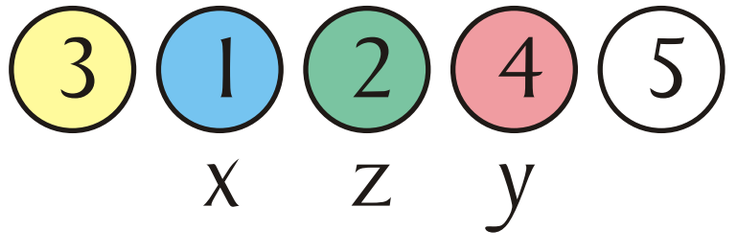
Generating Permutations by Prefix Rotations

Permutations of {1,2,...,n} by prefix rotations Corbett, Jiang-Ruskey, Ruskey-W.

Leftmost increase $x < y$

If $z > x$ then xzy

If $z < x$ then xyz



Generating Permutations by Prefix Rotations

Permutations of {1,2,...,n} by prefix rotations Corbett, Jiang-Ruskey, Ruskey-W.

Leftmost increase $x < y$

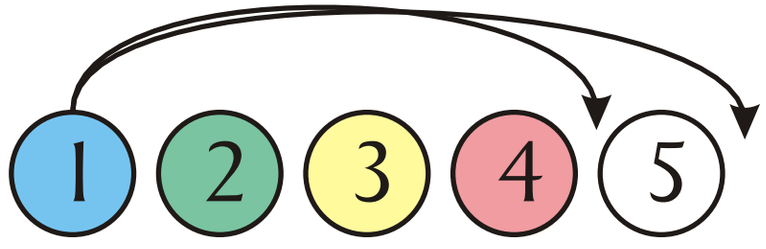
If $z > x$ then xzy

If $z < x$ then xyz

"Cool-lex"

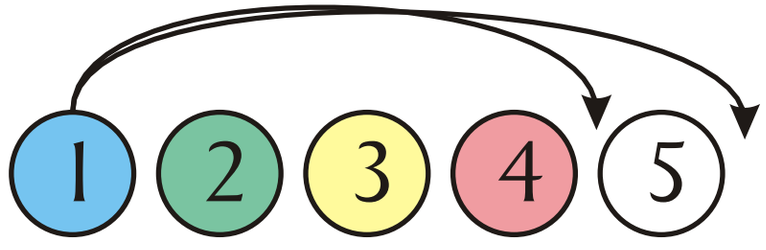
Question

Gray code using rotations of length n-1 and n?



Question

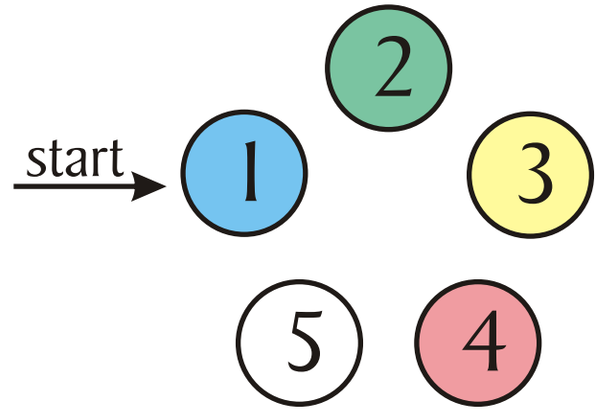
Gray code using rotations of length n-1 and n? Why?



Question

Gray code using rotations of length n-1 and n? Why?

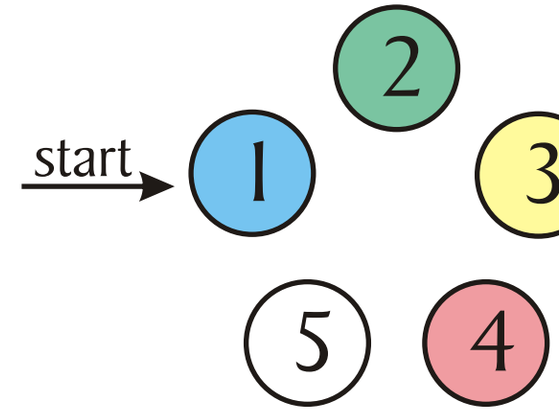
Efficient in circular arrays and linked list



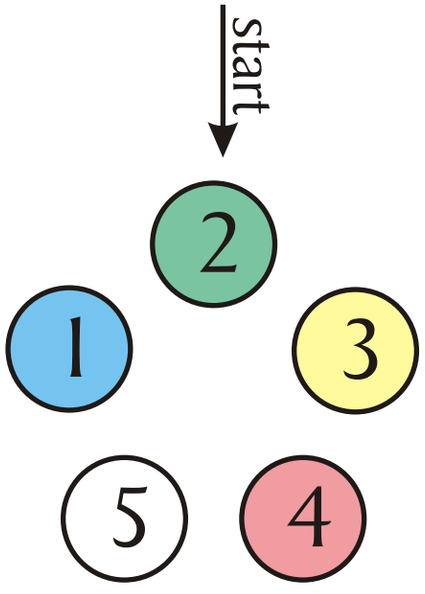
Question

Gray code using rotations of length n-1 and n? Why?

Efficient in circular arrays and linked list

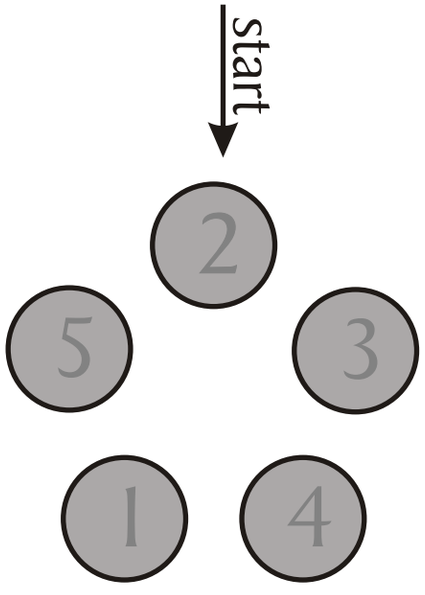


12345



23451

rotation of length 5



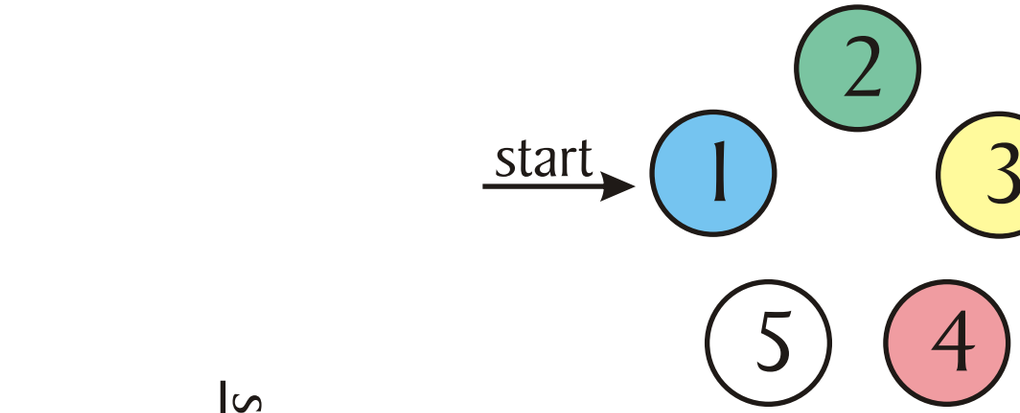
23415

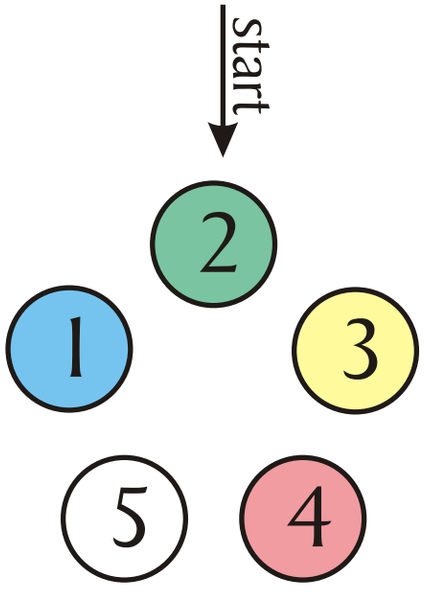
rotation of length 4

Question

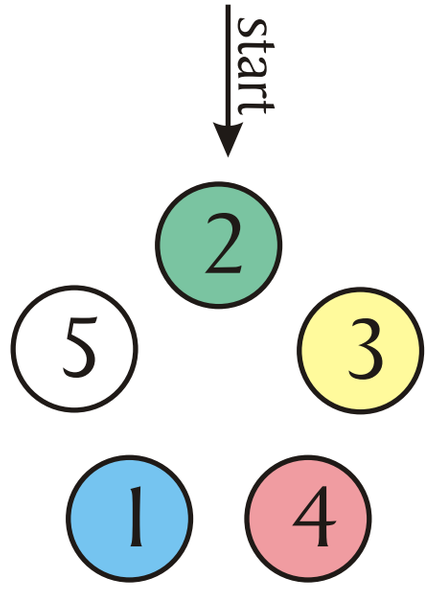
Gray code using rotations of length n-1 and n? Why?

Efficient in circular arrays and linked list





rotation of length 5

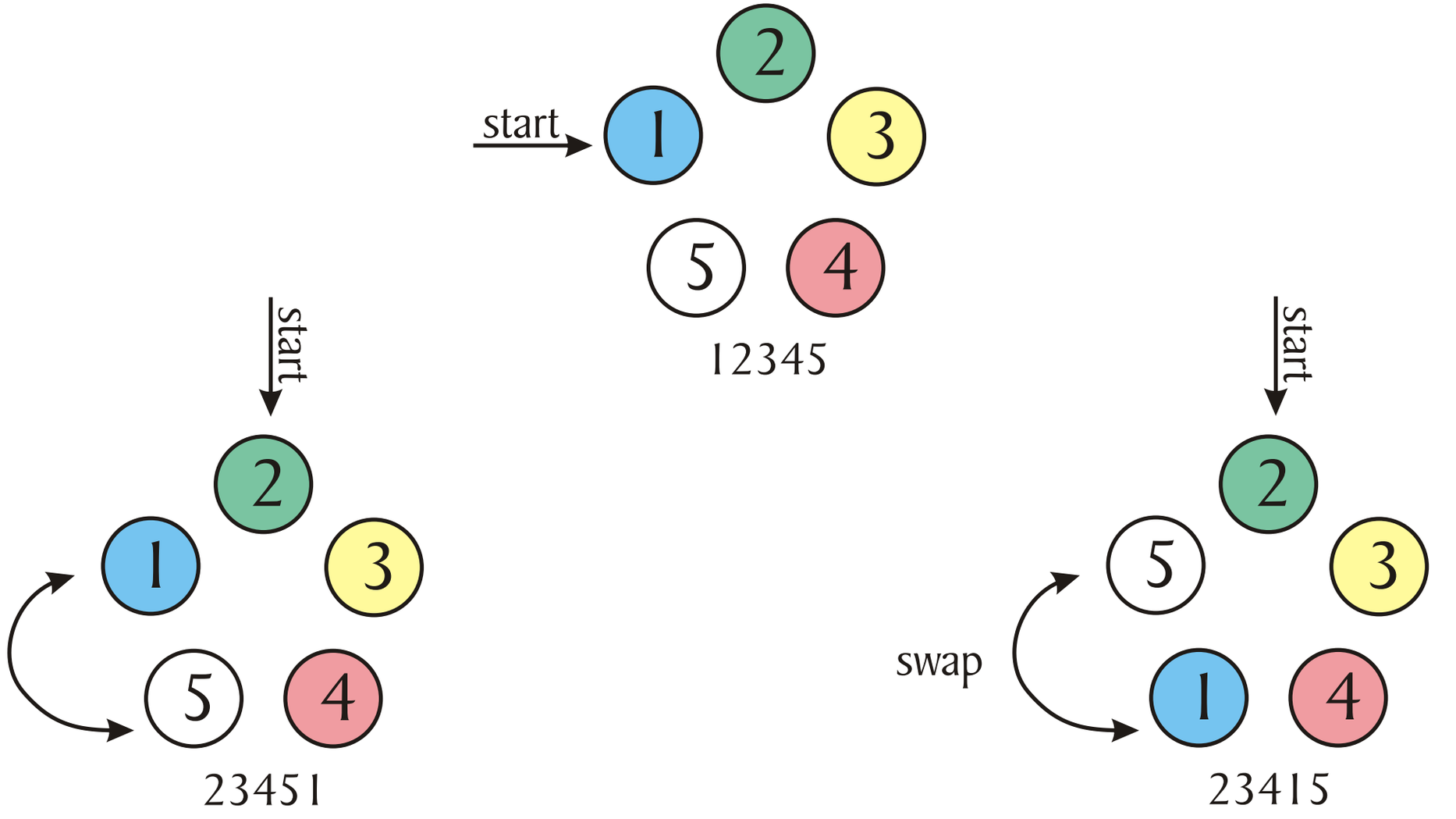


rotation of length 4

Question

Gray code using rotations of length n-1 and n? Why?

Efficient in circular arrays and linked list

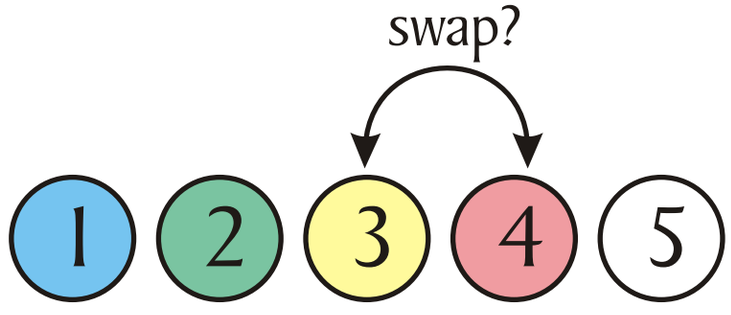


Plausibility

Is it possible to get from one permutation to another?

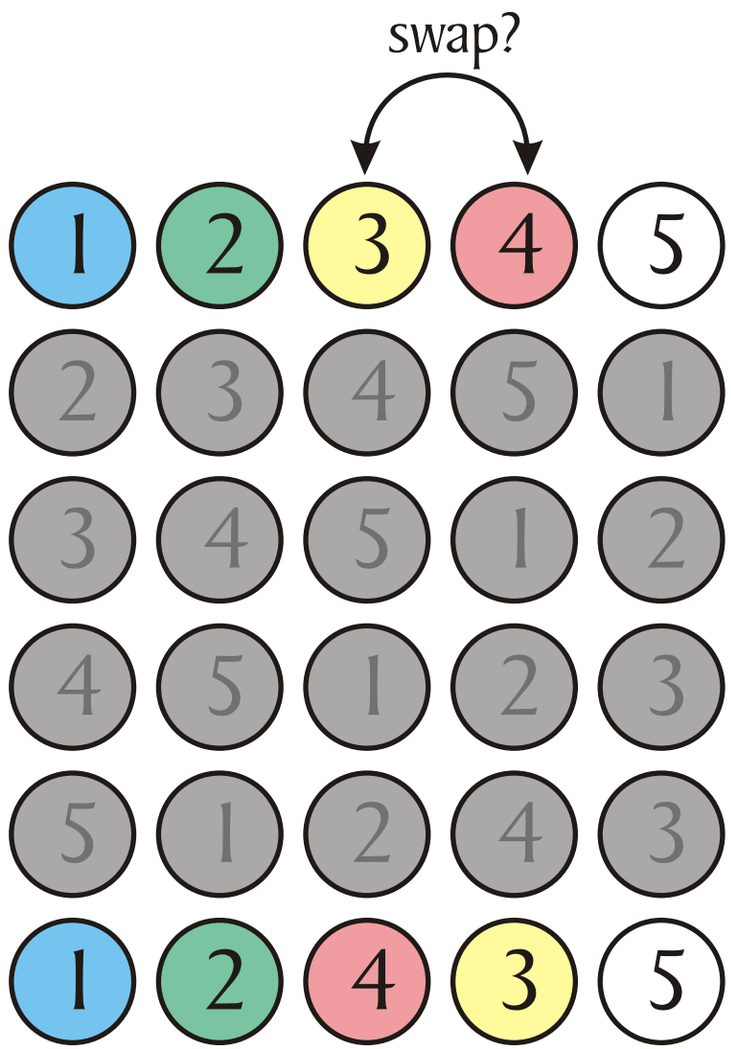
Plausibility

Is it possible to get from one permutation to another?



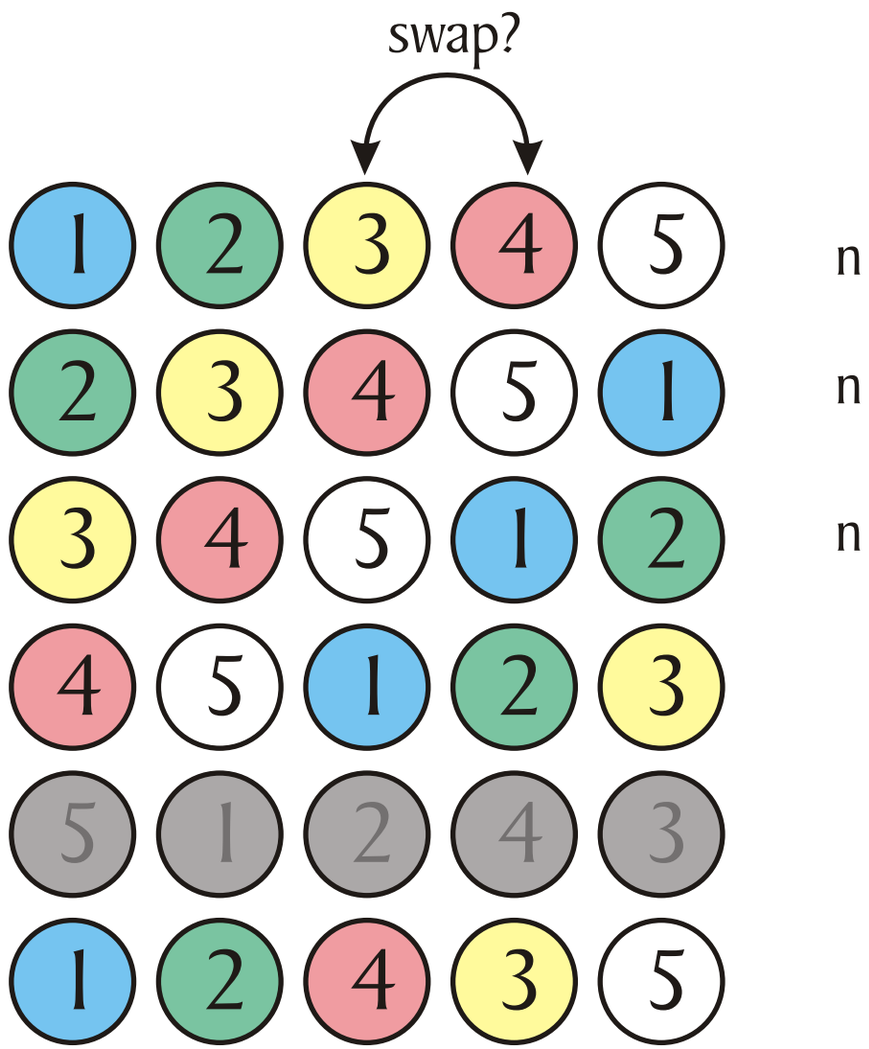
Plausibility

Is it possible to get from one permutation to another?



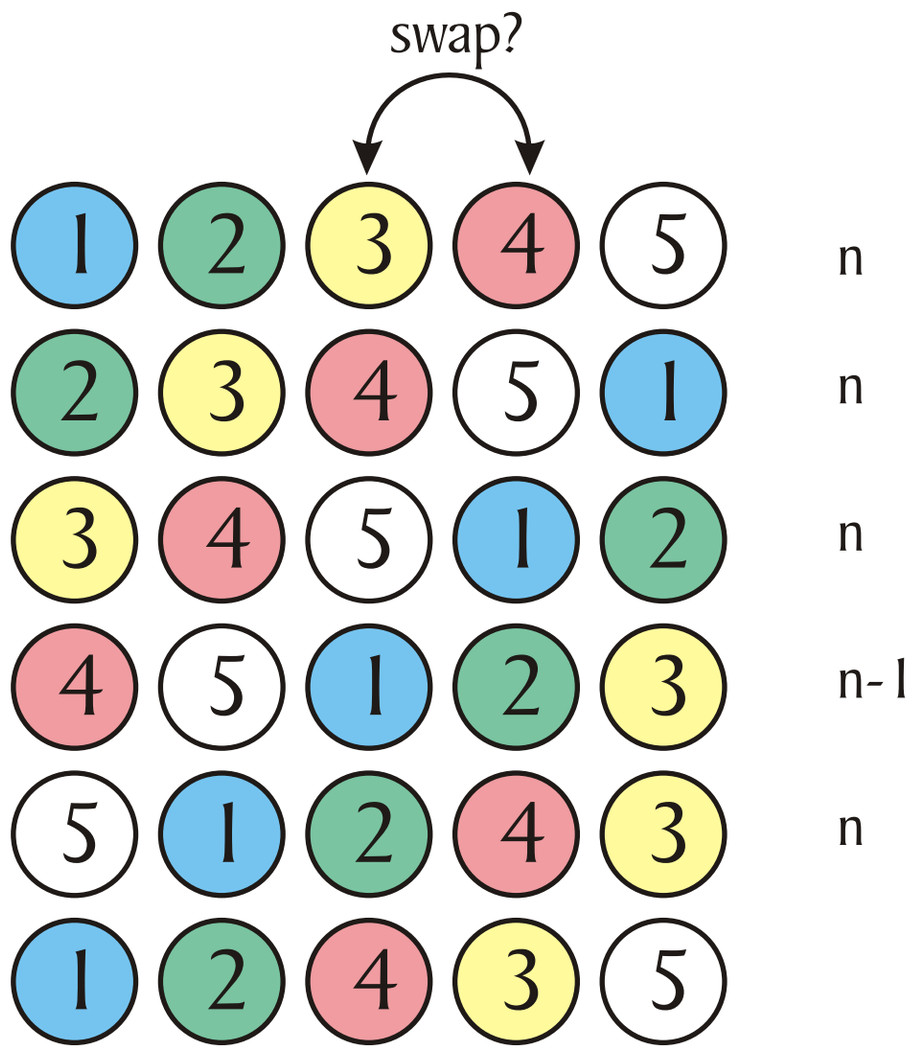
Plausibility

Is it possible to get from one permutation to another?

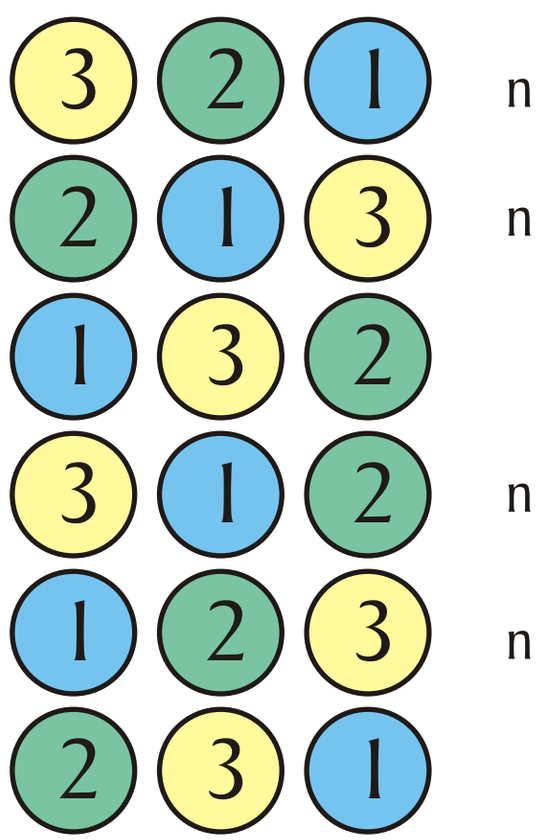


Plausibility

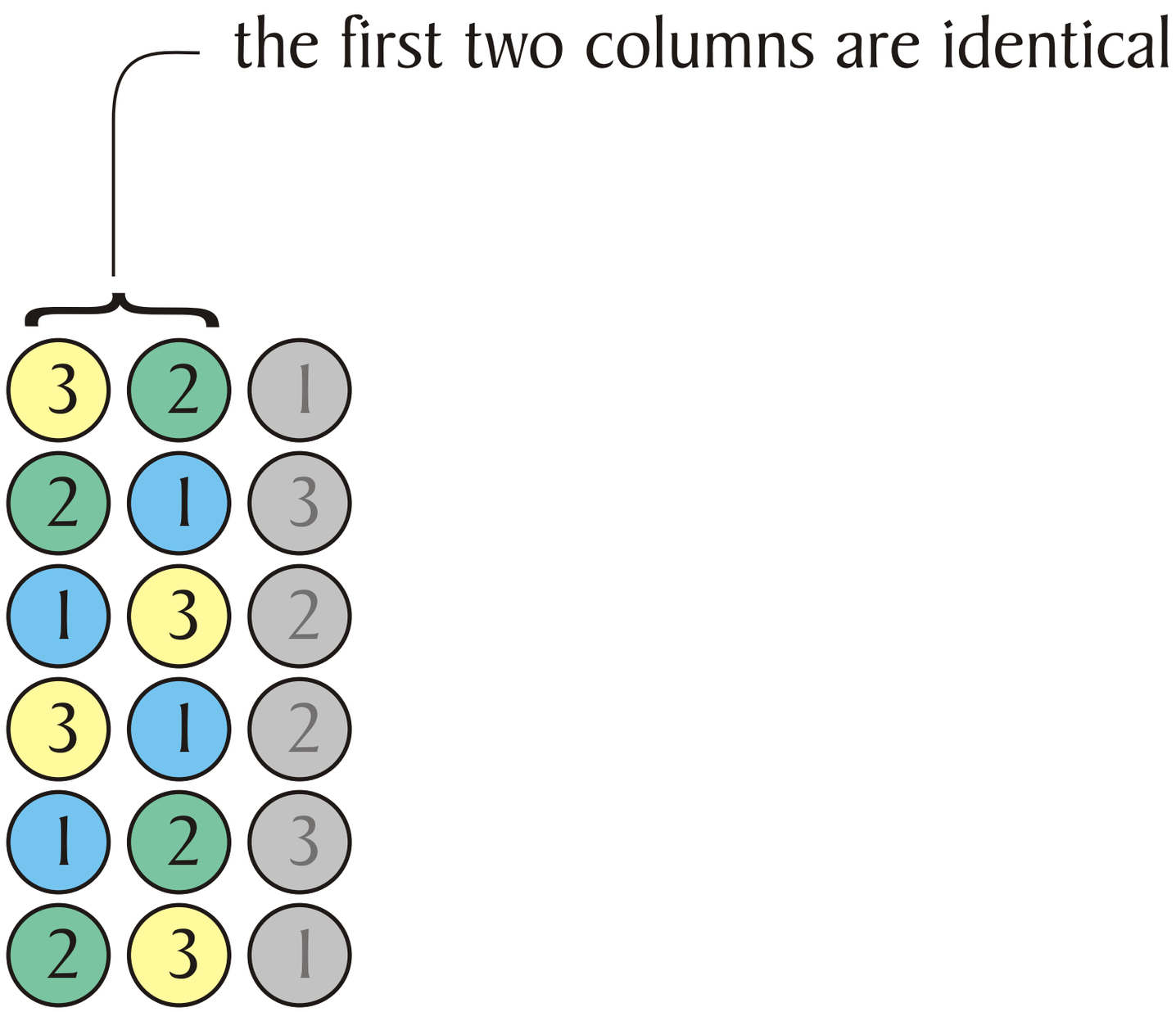
Is it possible to get from one permutation to another?



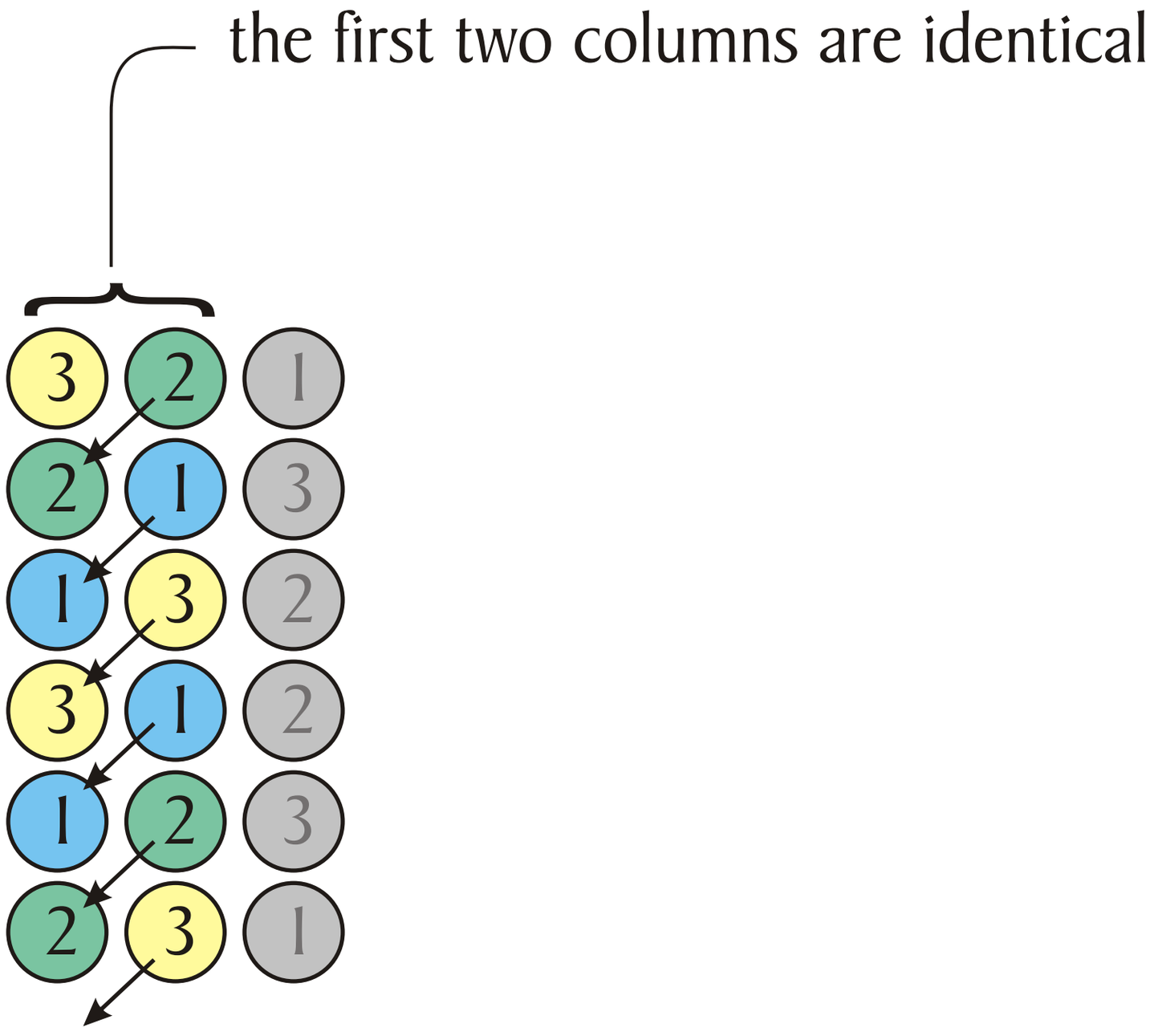
Observation



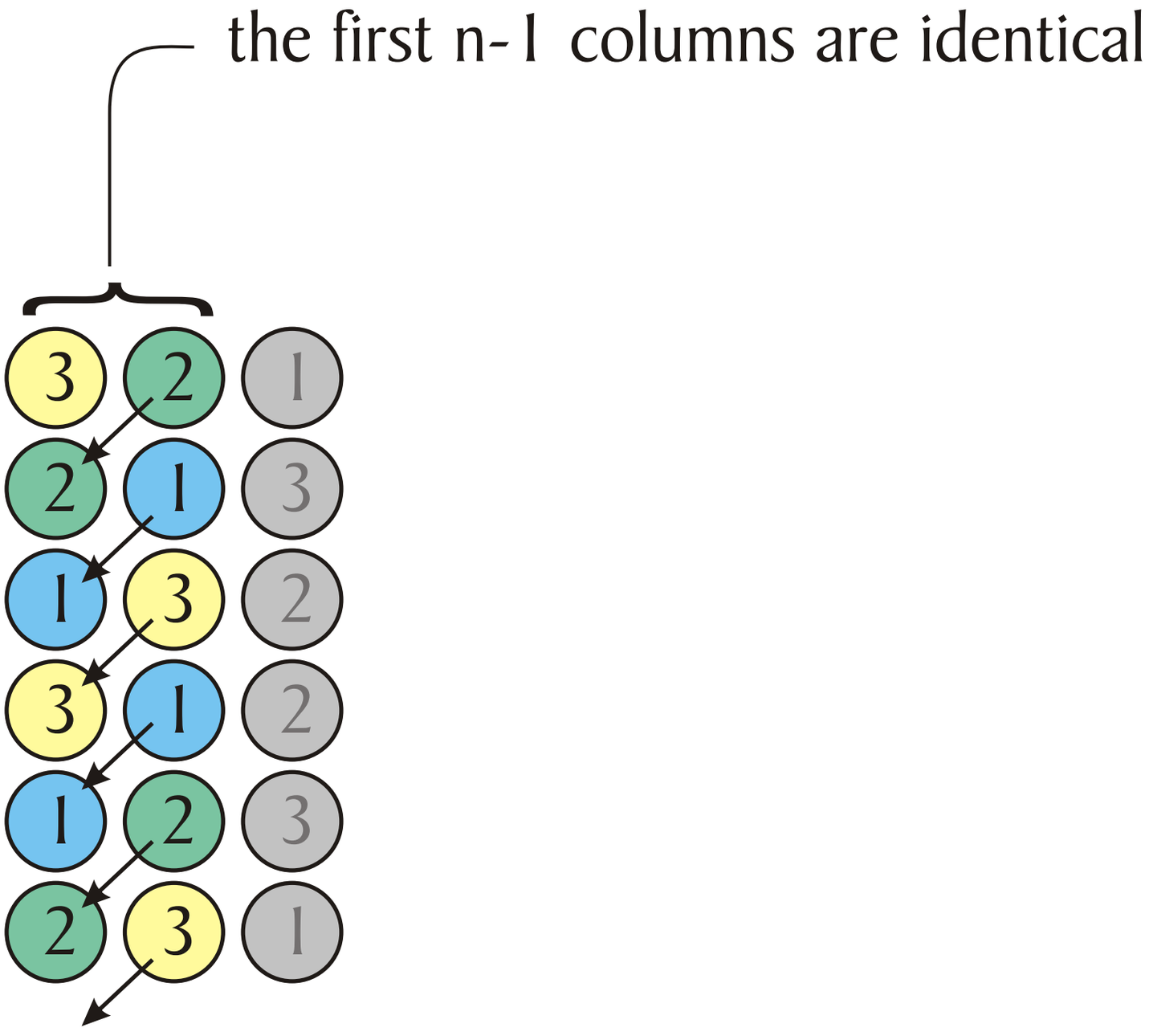
Observation



Observation



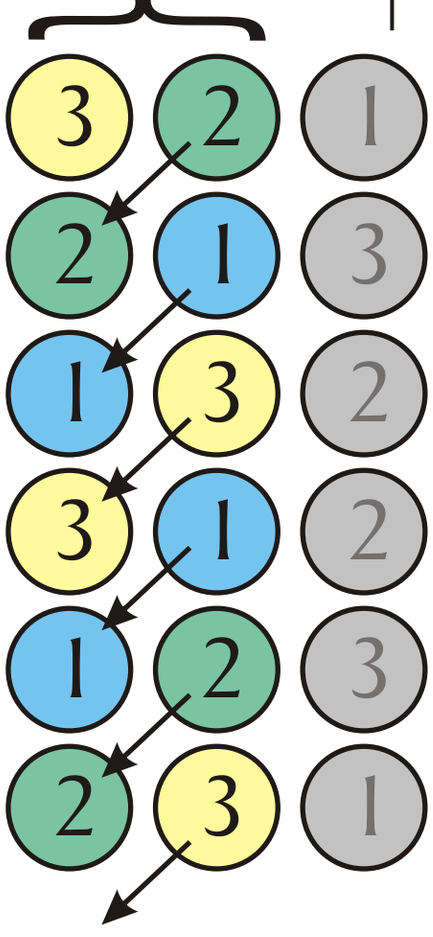
Observation



Observation

the first n-1 columns are identical

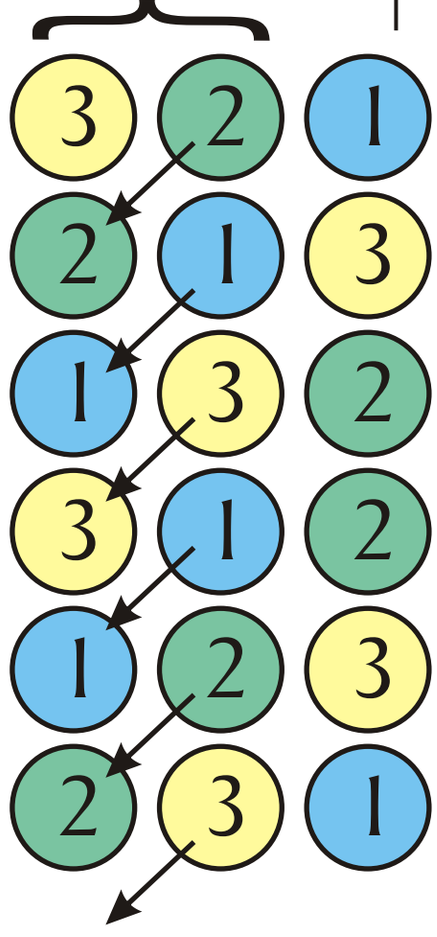
the last column is what is ‘missing’



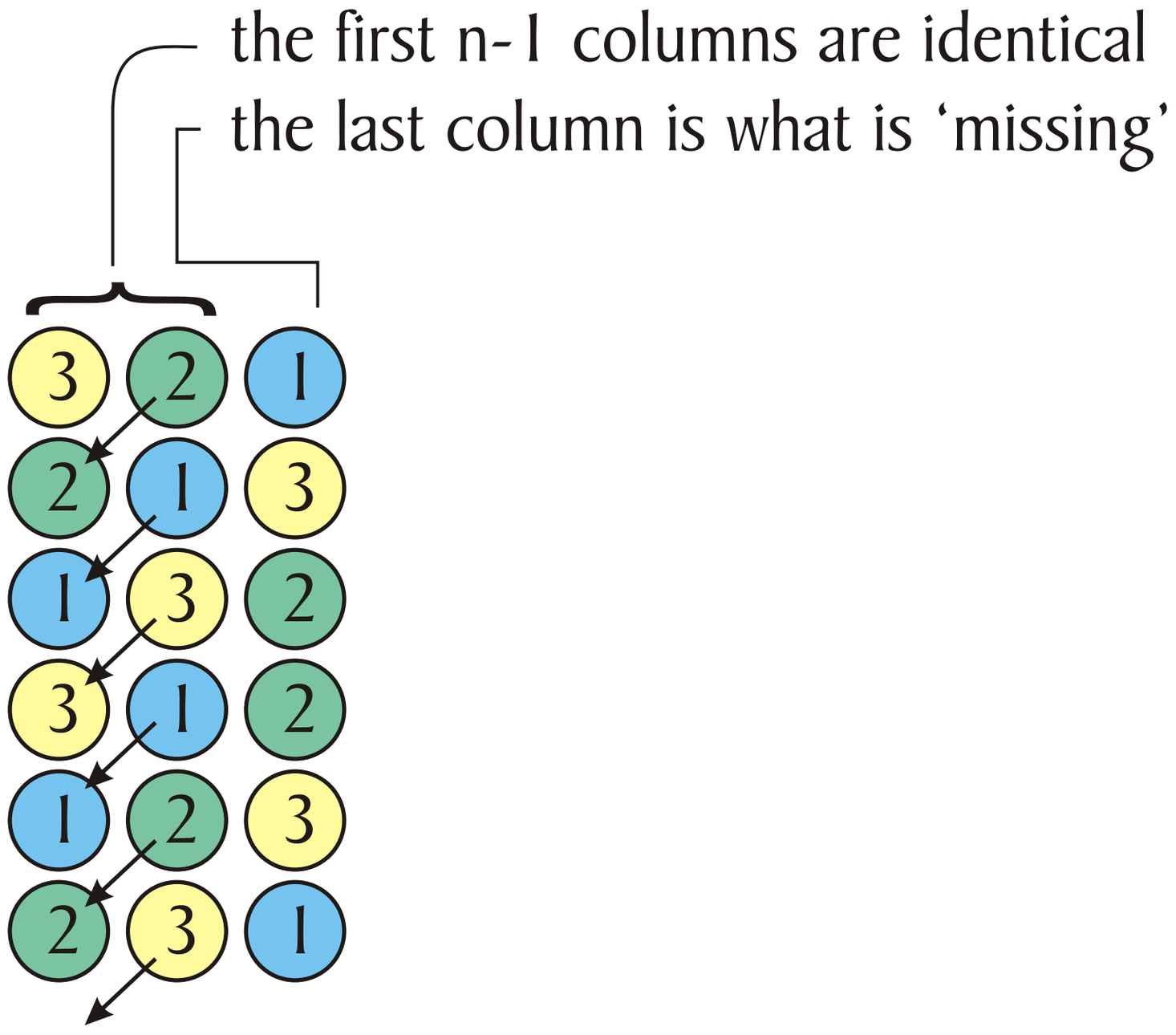
Observation

the first n-1 columns are identical

the last column is what is ‘missing’



Observation



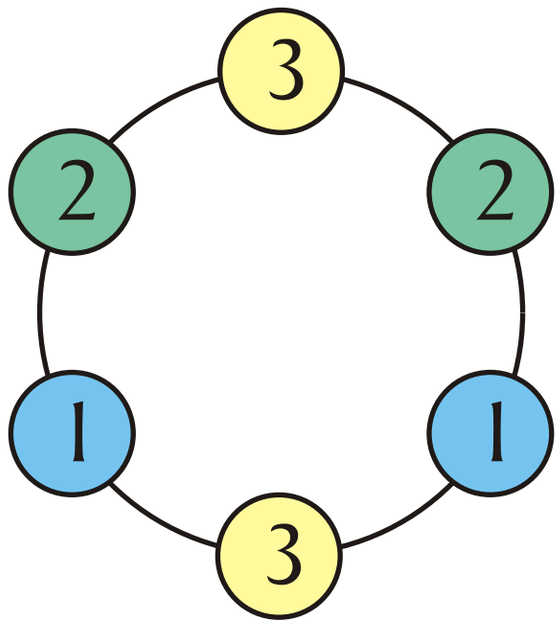
Gray code determined by one column

Shorthand Universal Cycles

Circular string using {1,2,...,n} of length n!

Distinct substrings of length n-1

Append missing symbol



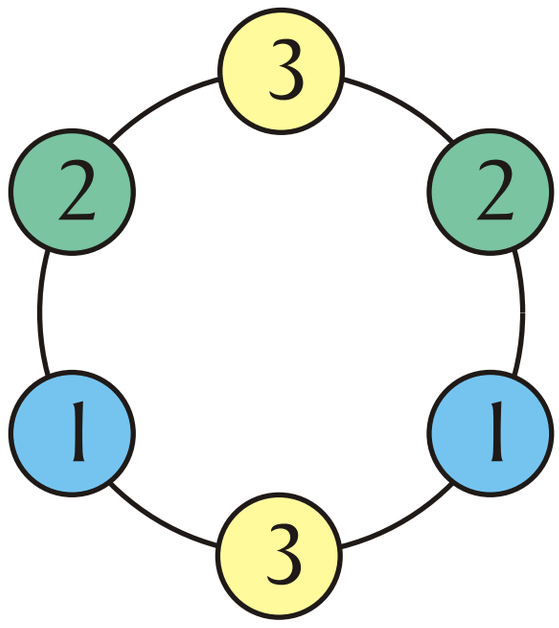
Shorthand Universal Cycles

Circular string using {1,2,...,n} of length n!

Distinct substrings of length n-1

Append missing symbol

} Shorthand for permutation



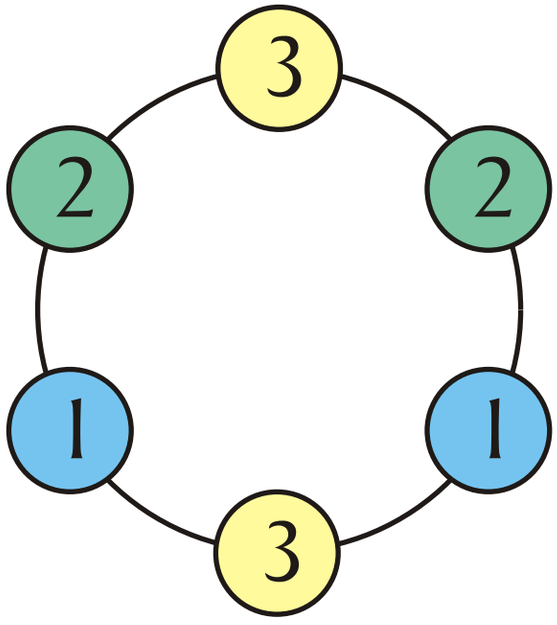
Shorthand Universal Cycles

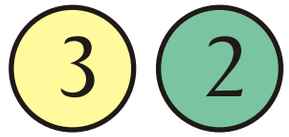
Circular string using {1,2,...,n} of length n!

Distinct substrings of length n-1

Append missing symbol

} Shorthand for permutation





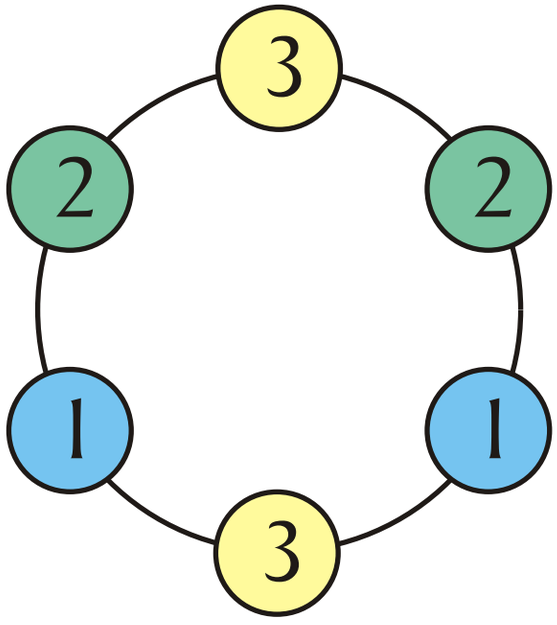
Shorthand Universal Cycles

Circular string using {1,2,...,n} of length n!

Distinct substrings of length n-1

Append missing symbol

} Shorthand for permutation



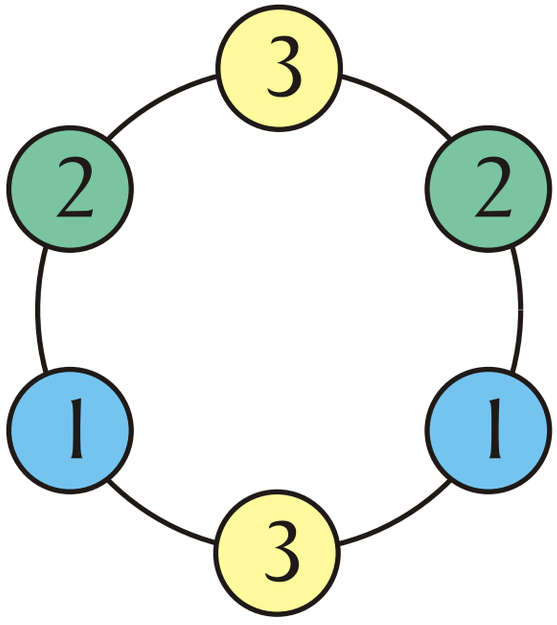
Shorthand Universal Cycles

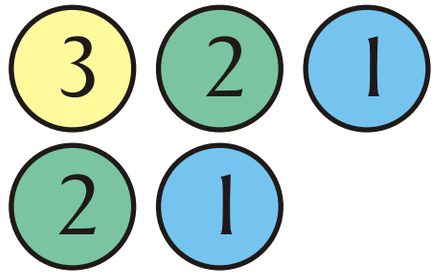
Circular string using {1,2,...,n} of length n!

Distinct substrings of length n-1

Append missing symbol

} Shorthand for permutation





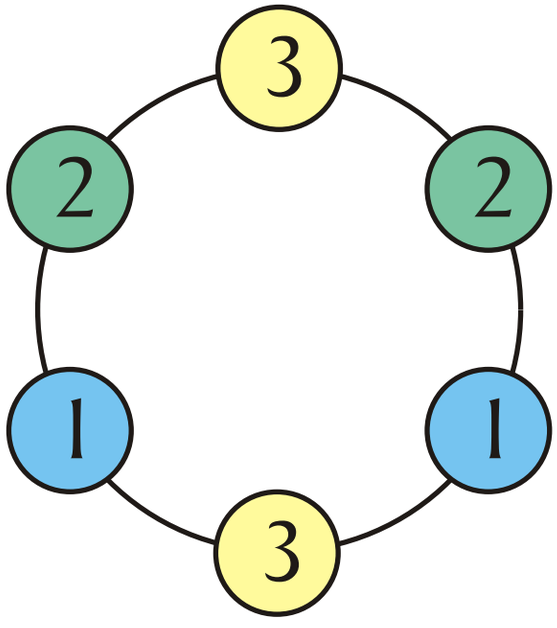
Shorthand Universal Cycles

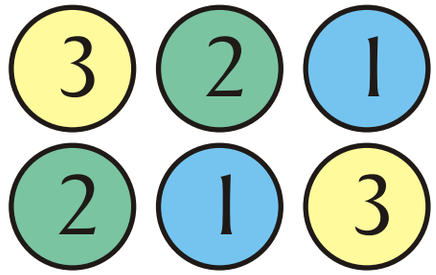
Circular string using {1,2,...,n} of length n!

Distinct substrings of length n-1

Append missing symbol

} Shorthand for permutation





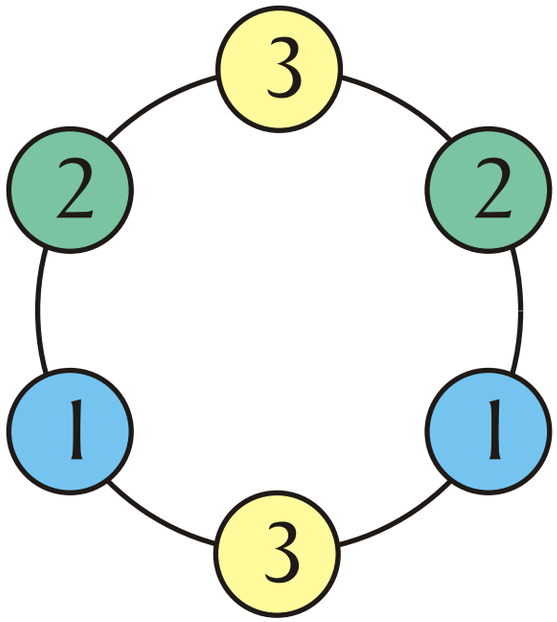
Shorthand Universal Cycles

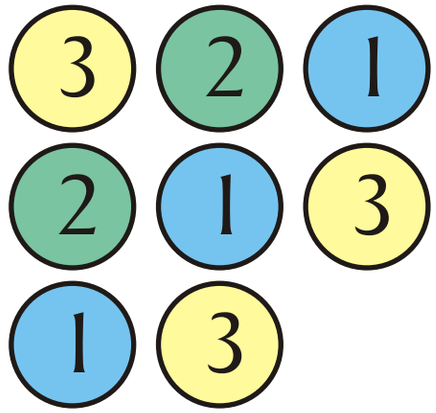
Circular string using {1,2,...,n} of length n!

Distinct substrings of length n-1

Append missing symbol

} Shorthand for permutation





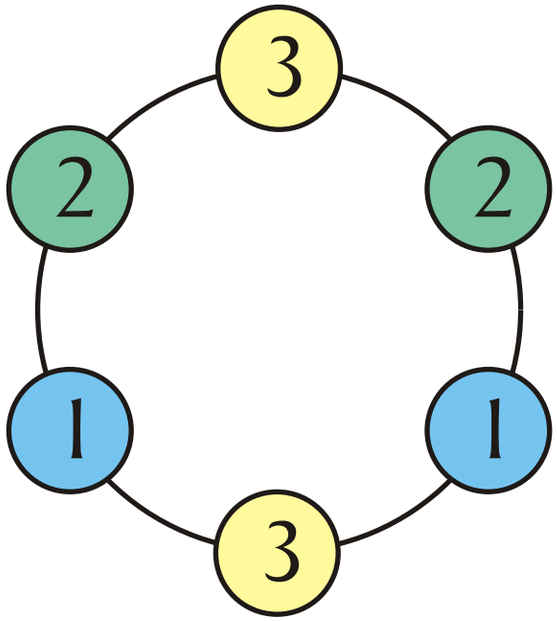
Shorthand Universal Cycles

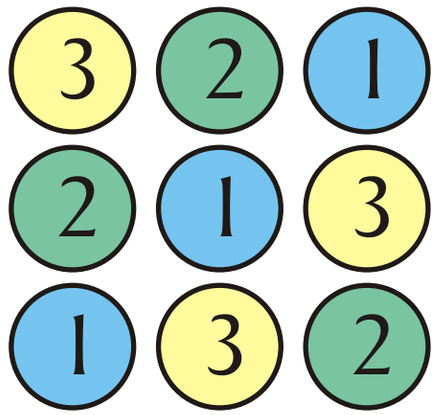
Circular string using {1,2,...,n} of length n!

Distinct substrings of length n-1

Append missing symbol

} Shorthand for permutation





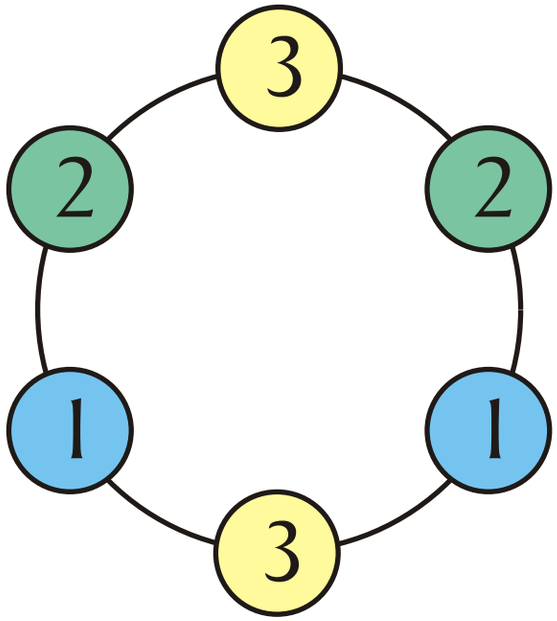
Shorthand Universal Cycles

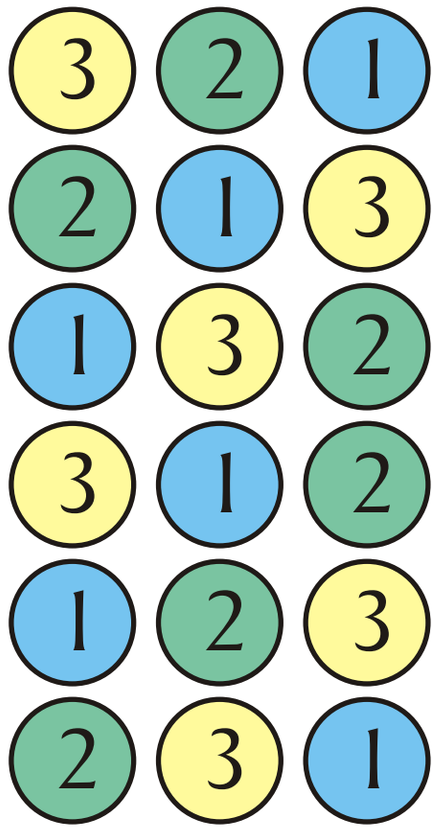
Circular string using {1,2,...,n} of length n!

Distinct substrings of length n-1

Append missing symbol

} Shorthand for permutation





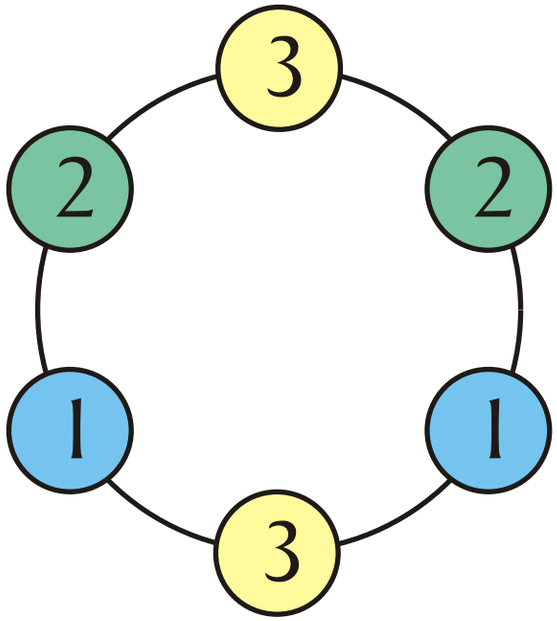
Shorthand Universal Cycles

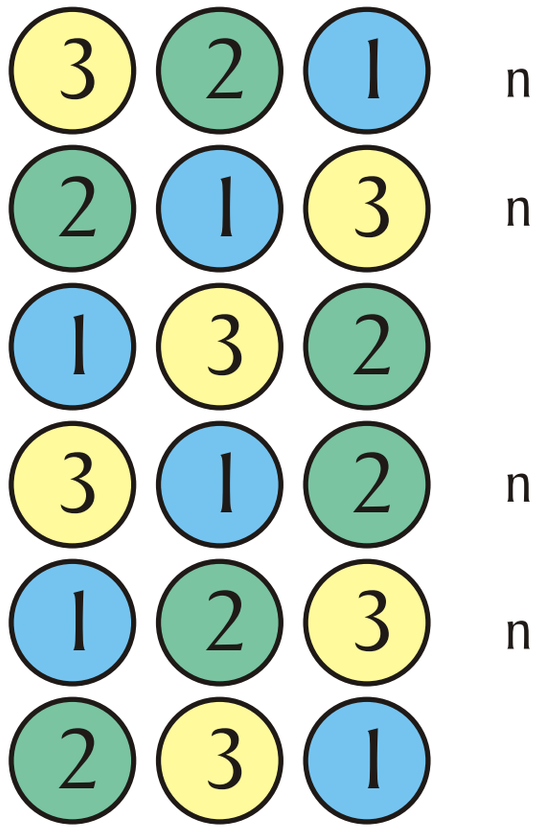
Circular string using {1,2,...,n} of length n!

Distinct substrings of length n-1

Append missing symbol

} Shorthand for permutation





Shorthand Universal Cycles ⇒ Gray codes

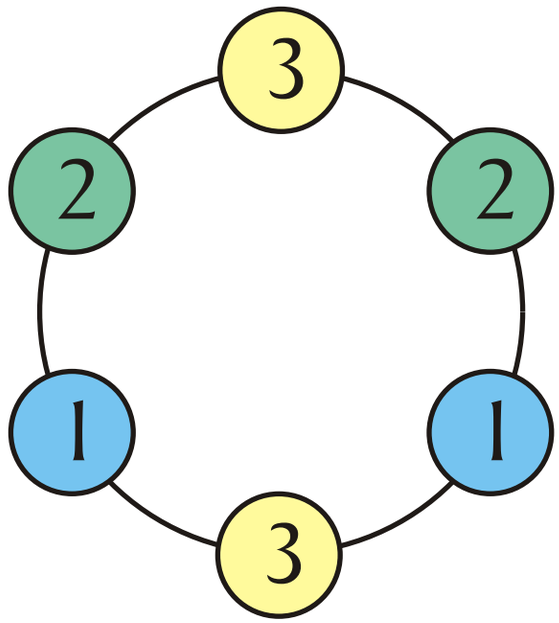
Shorthand Universal Cycles

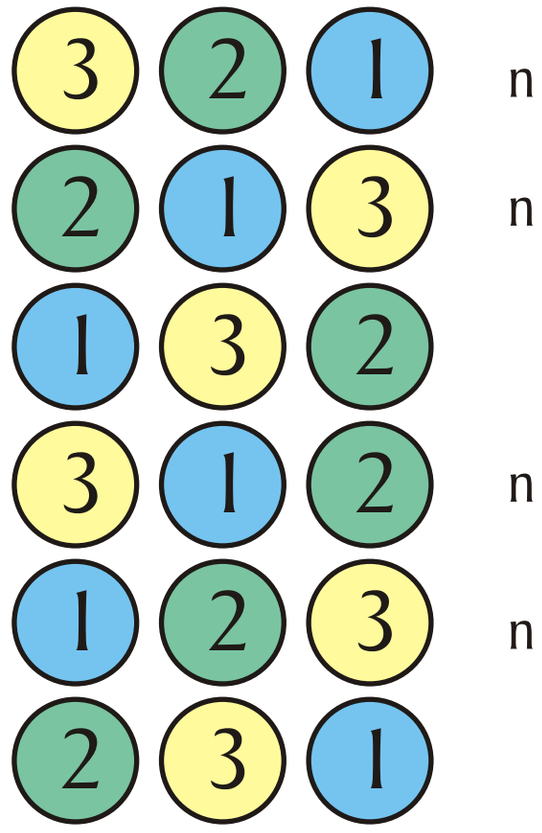
Circular string using {1,2,...,n} of length n!

Distinct substrings of length n-1

Append missing symbol

} Shorthand for permutation



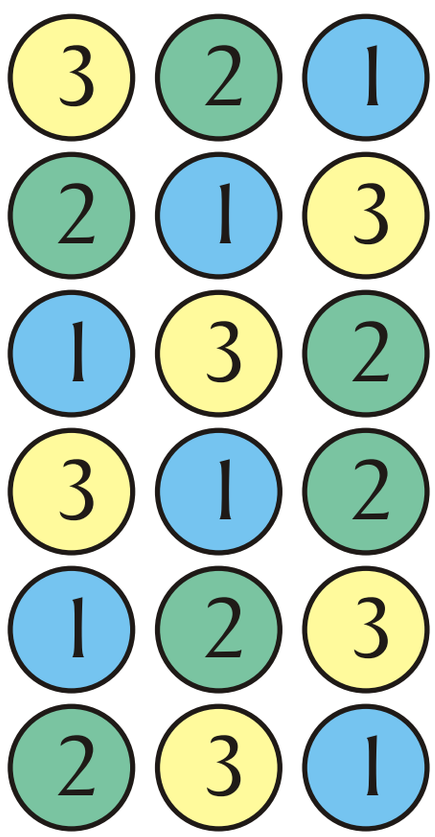


Column of Gray code ⇒ Shorthand Universal Cycle

Construction

Construction

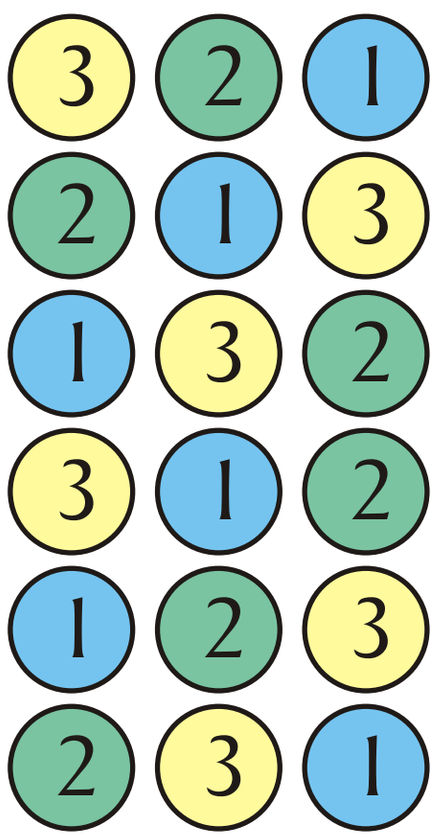
Gray code for n



Construction

Gray code for n

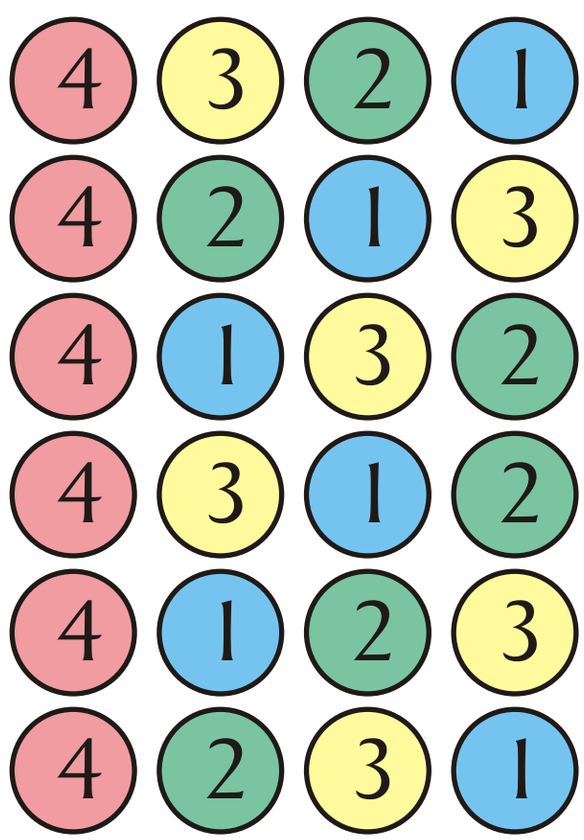
Prefix each row with n+1



Construction

Gray code for n

Prefix each row with n+1

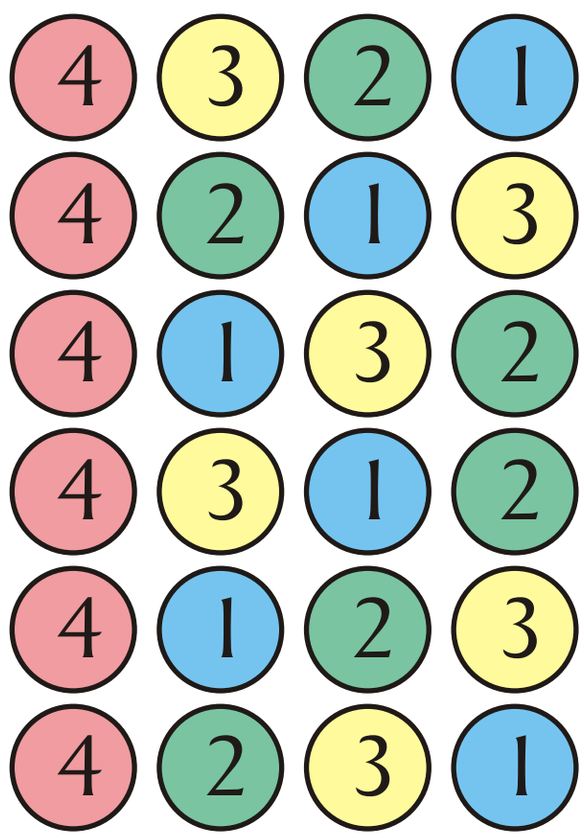


Construction

Gray code for n

Prefix each row with n+1

Shorthand Universal Cycle for n+1

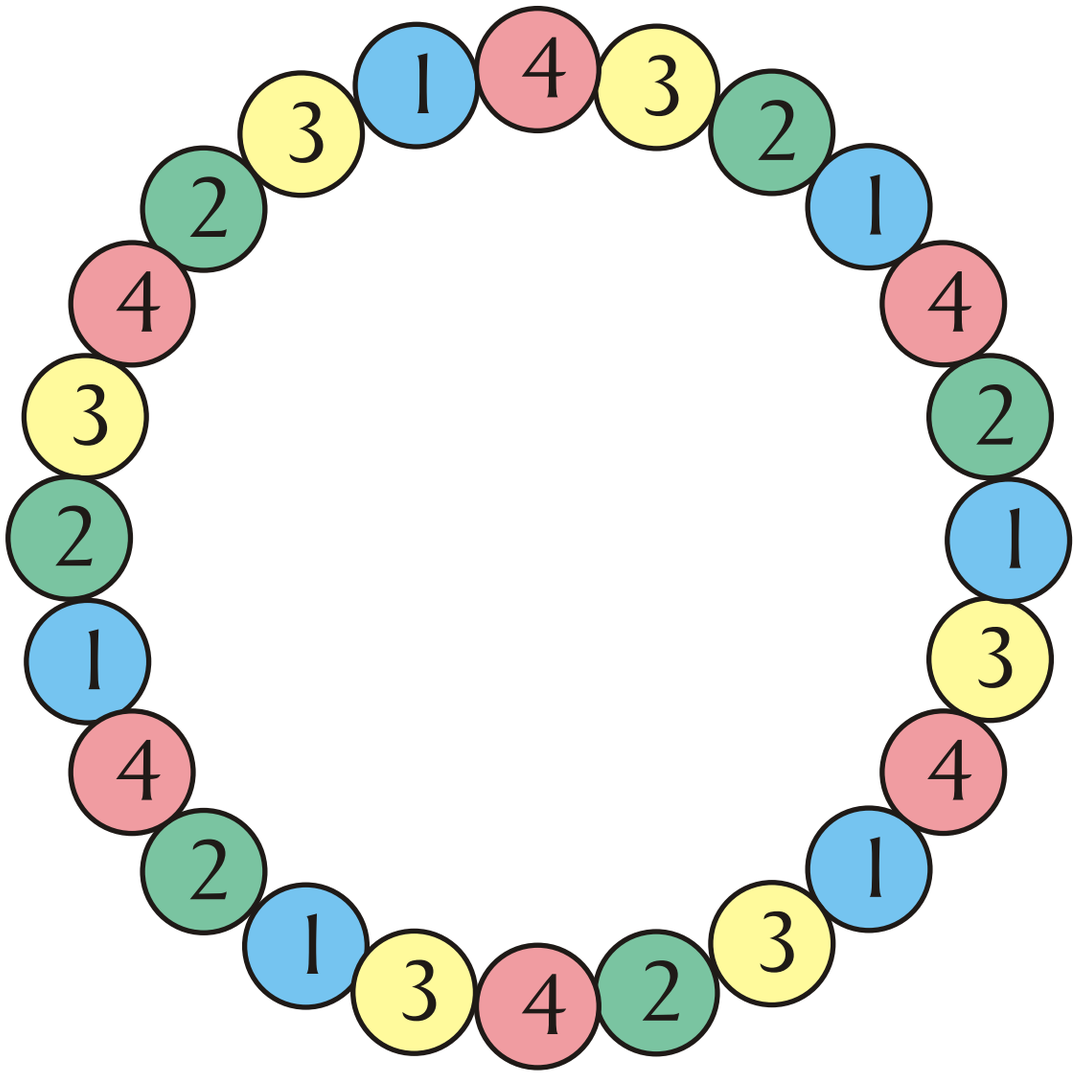


Construction

Gray code for n

Prefix each row with n+1

Shorthand Universal Cycle for n+1

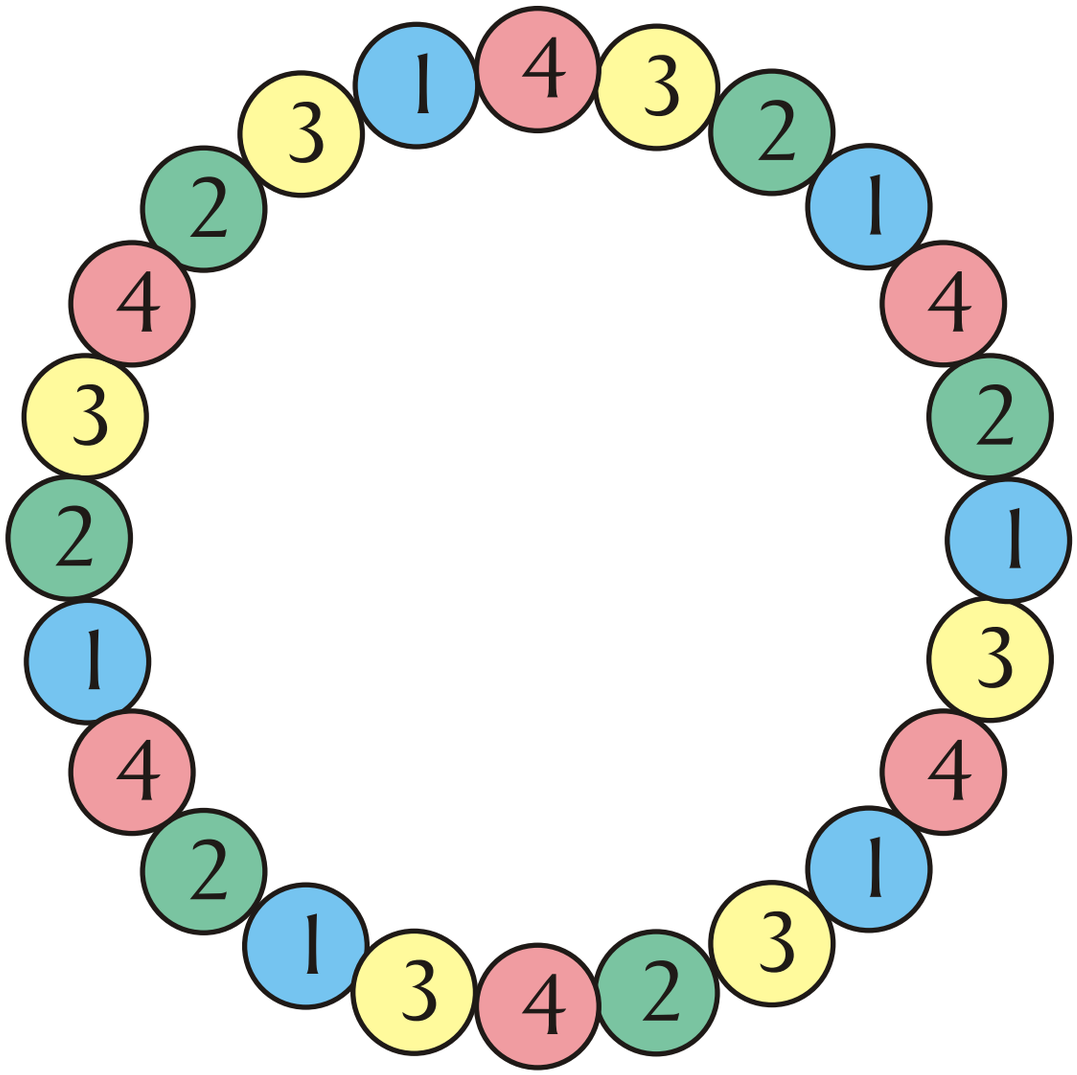


Construction

Gray code for n

Prefix each row with n+1

Shorthand Universal Cycle for n+1



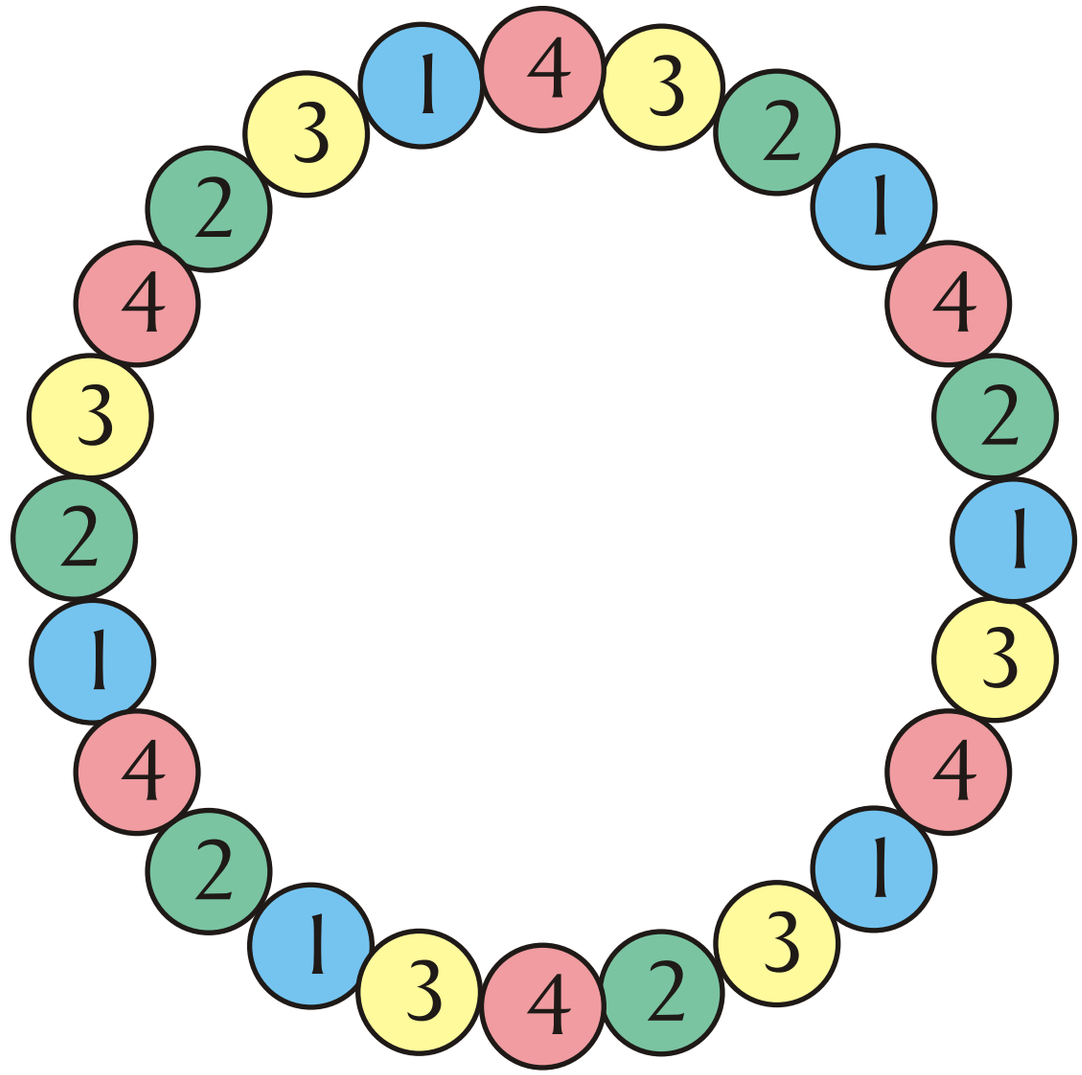
Why?

Construction

Gray code for n

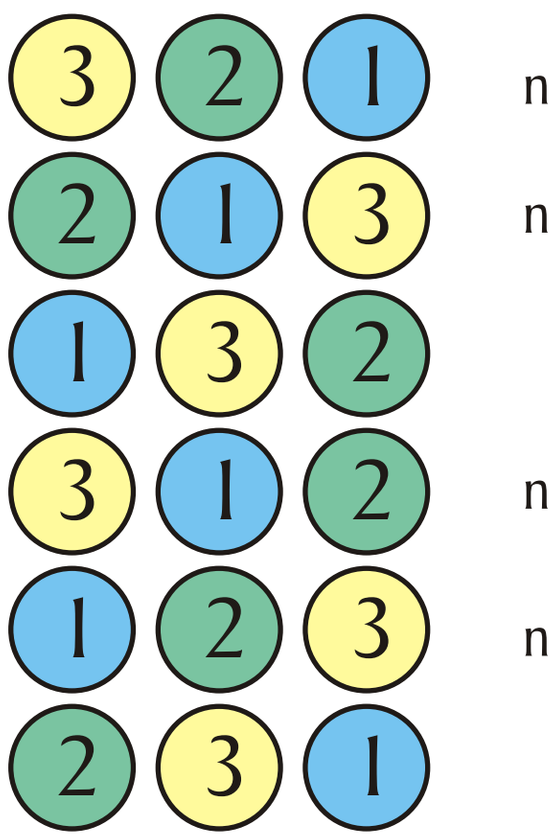
Prefix each row with n+1

Shorthand Universal Cycle for n+1

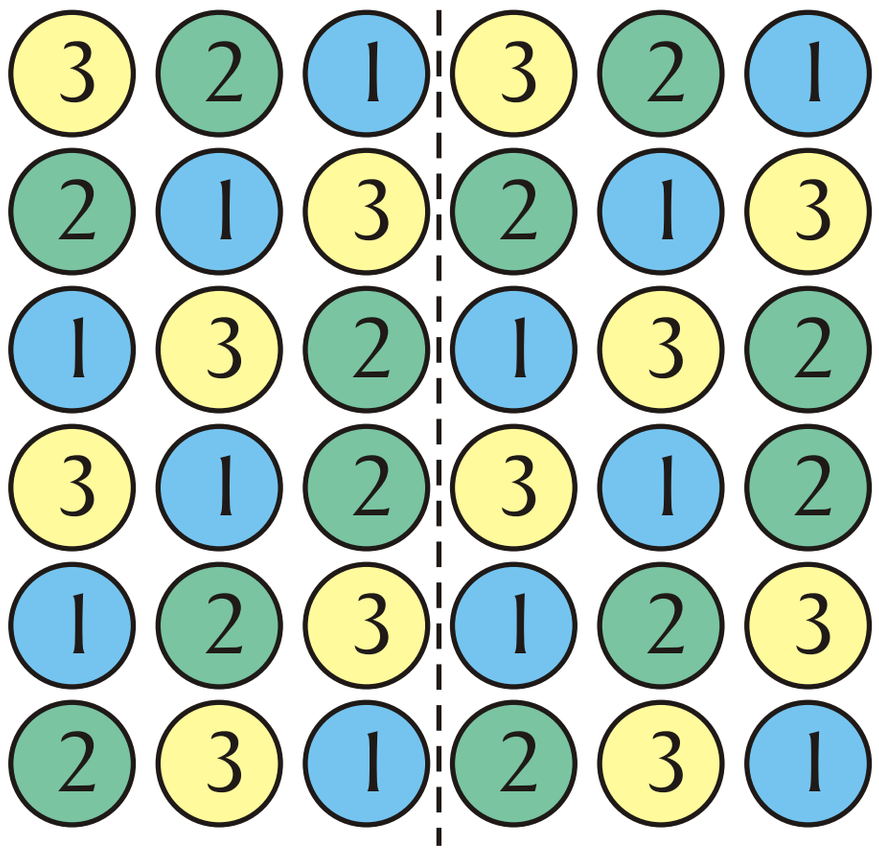


Correct size

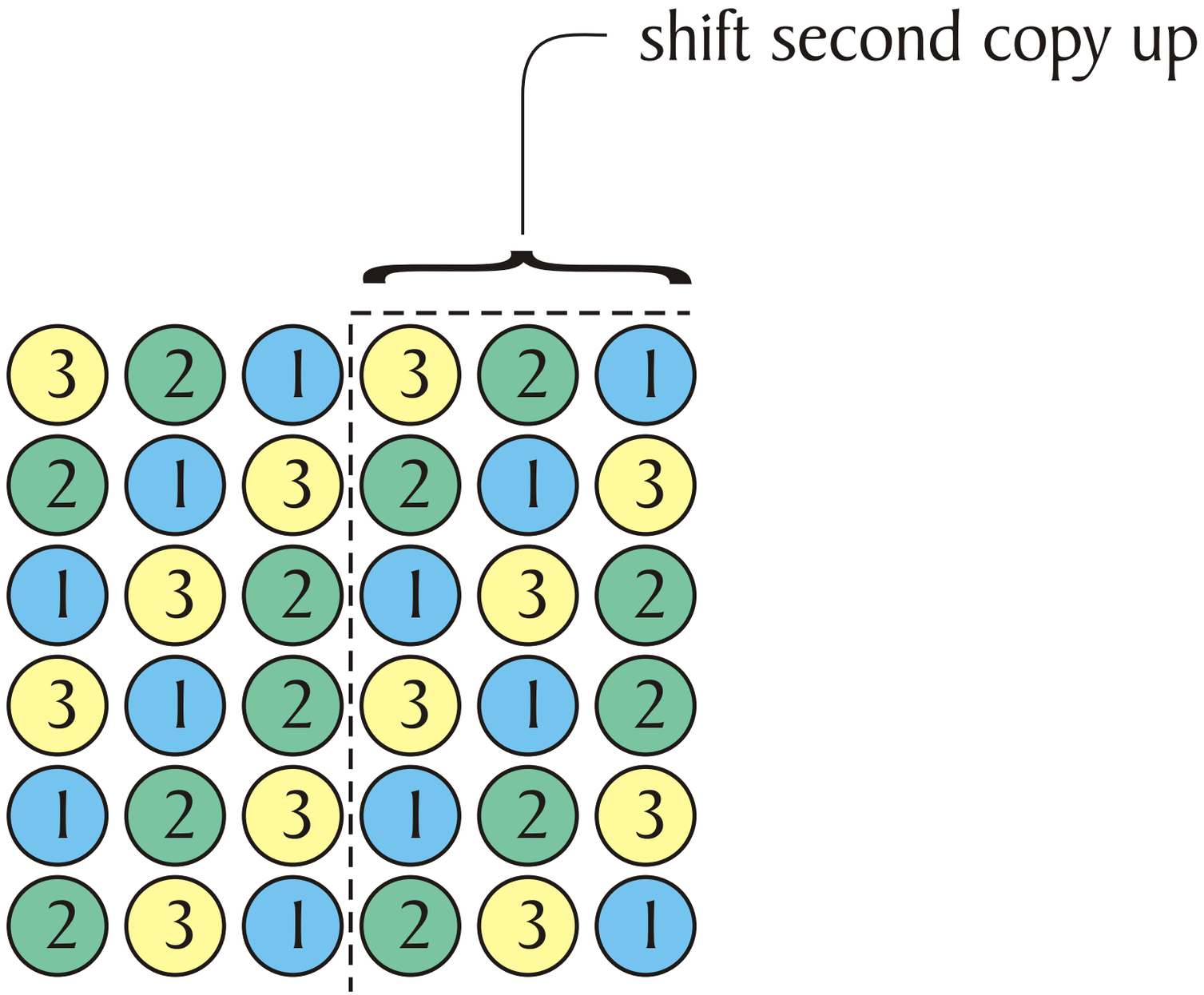
Circulant Property



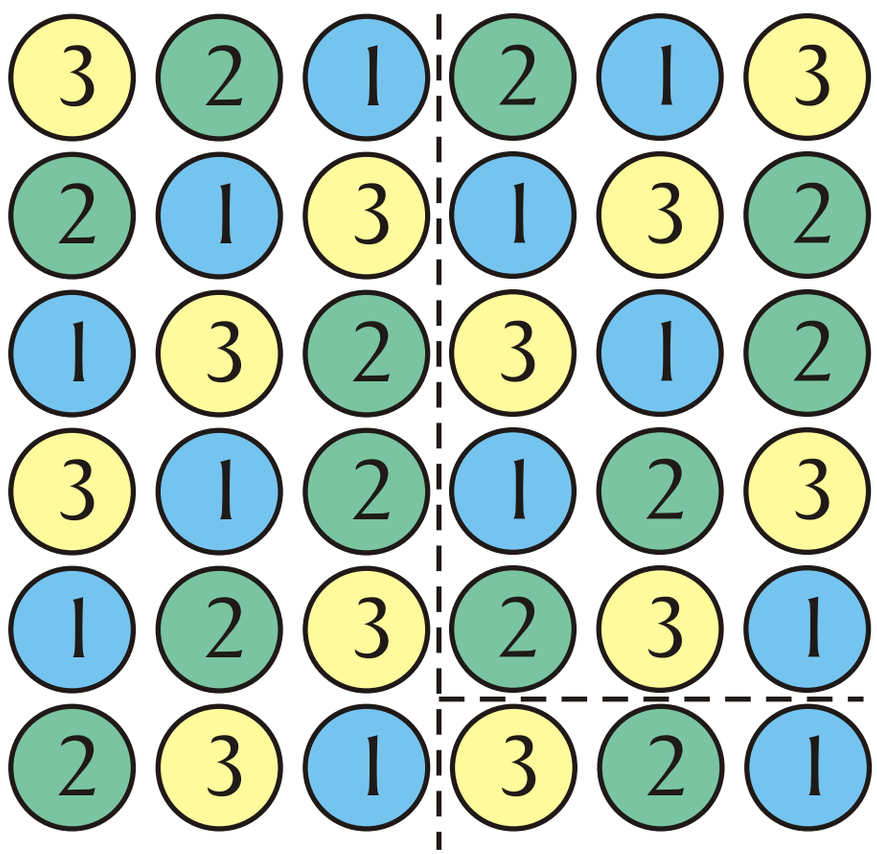
Circulant Property



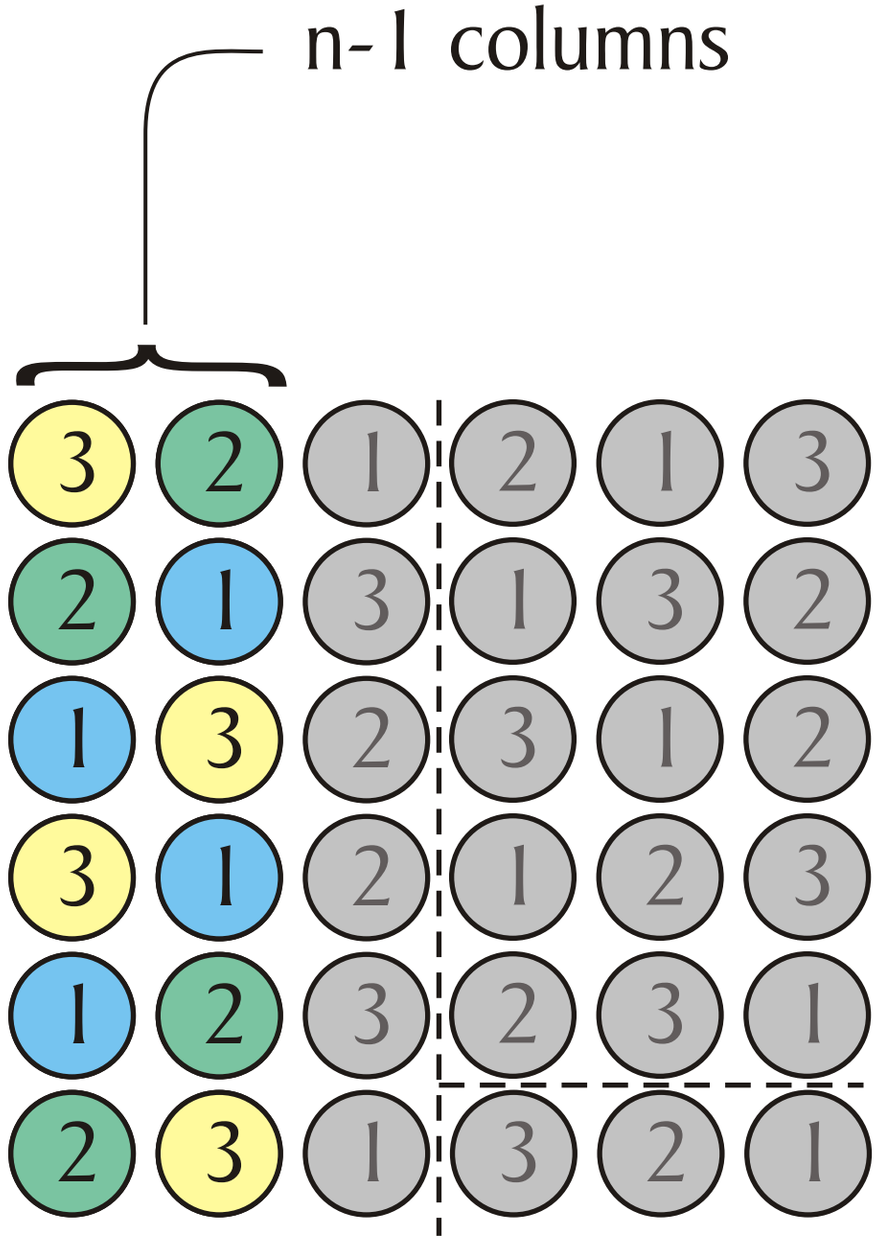
Circulant Property



Circulant Property

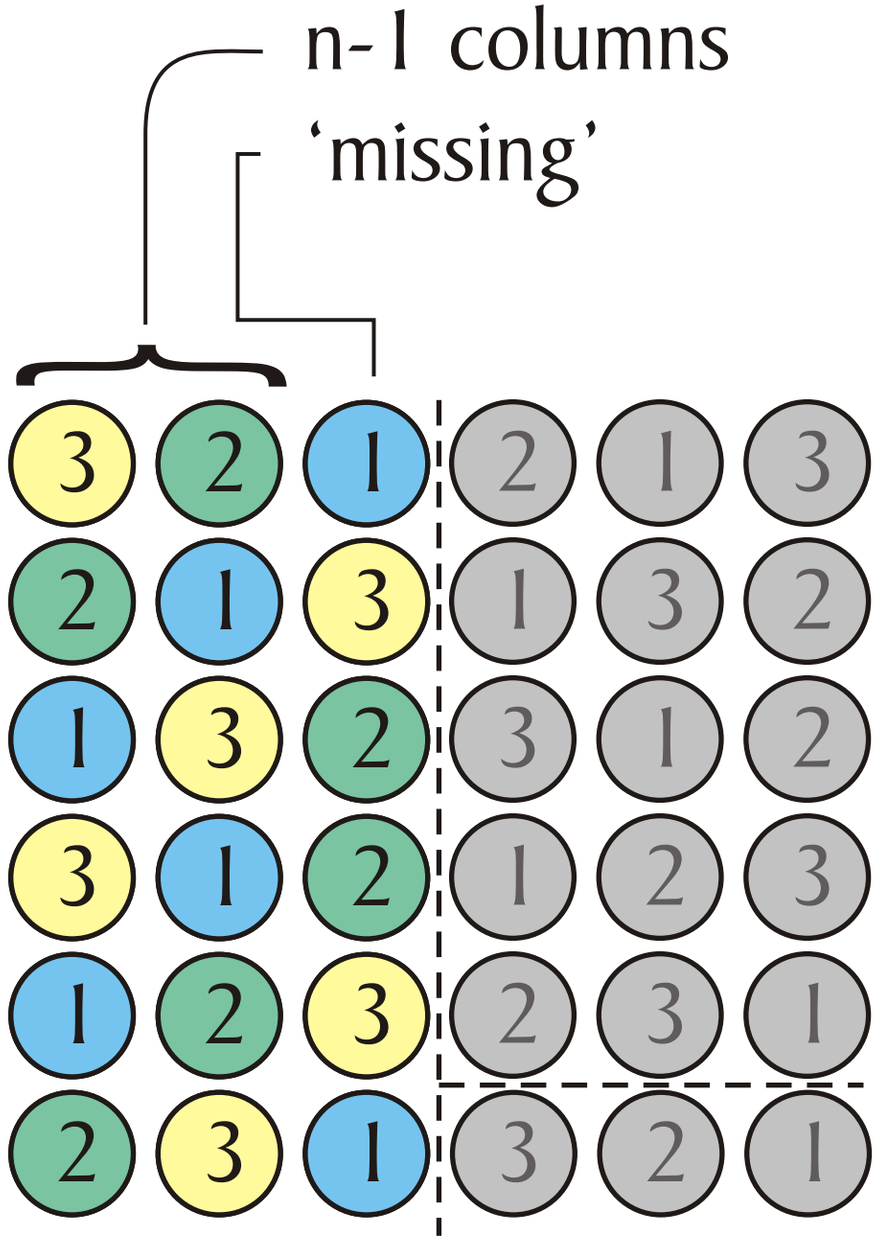


Circulant Property



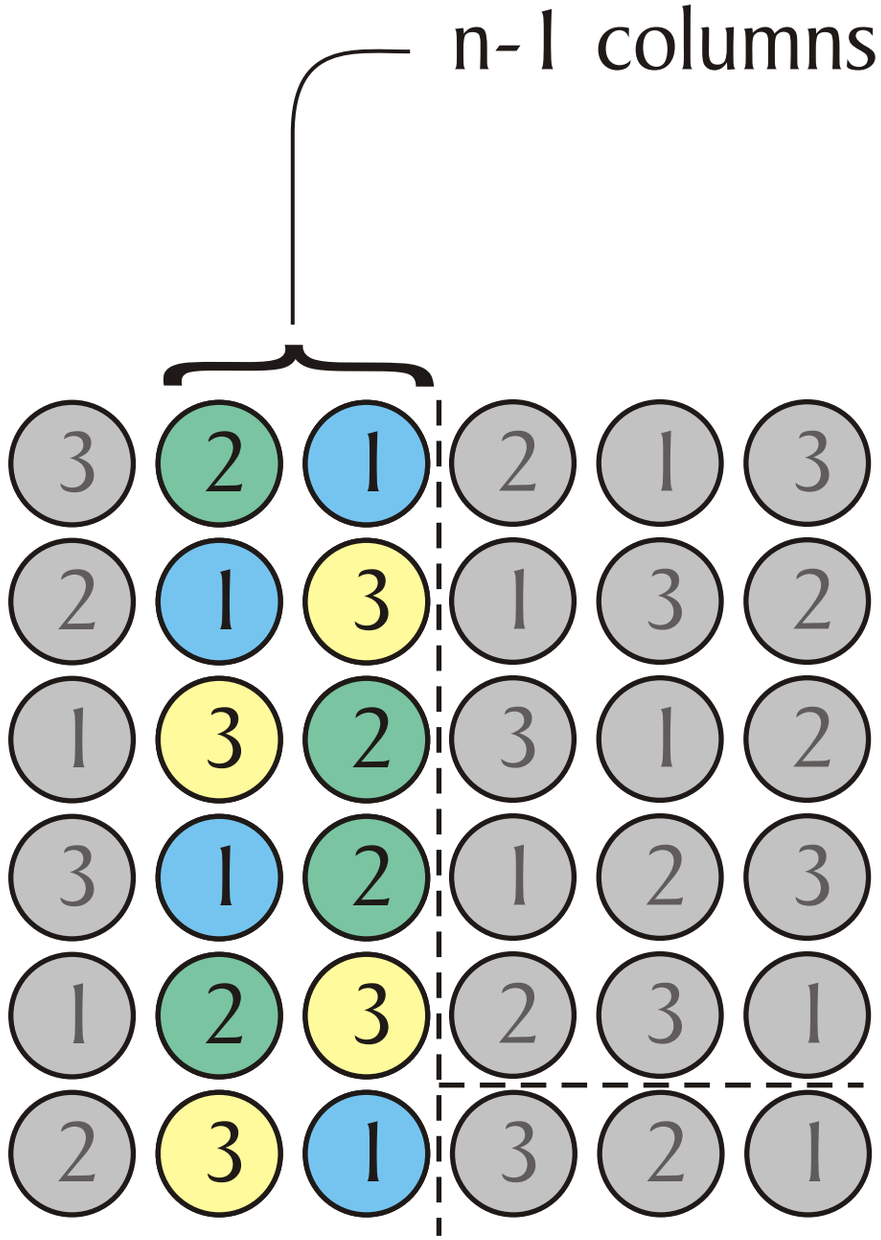
Offset 0

Circulant Property



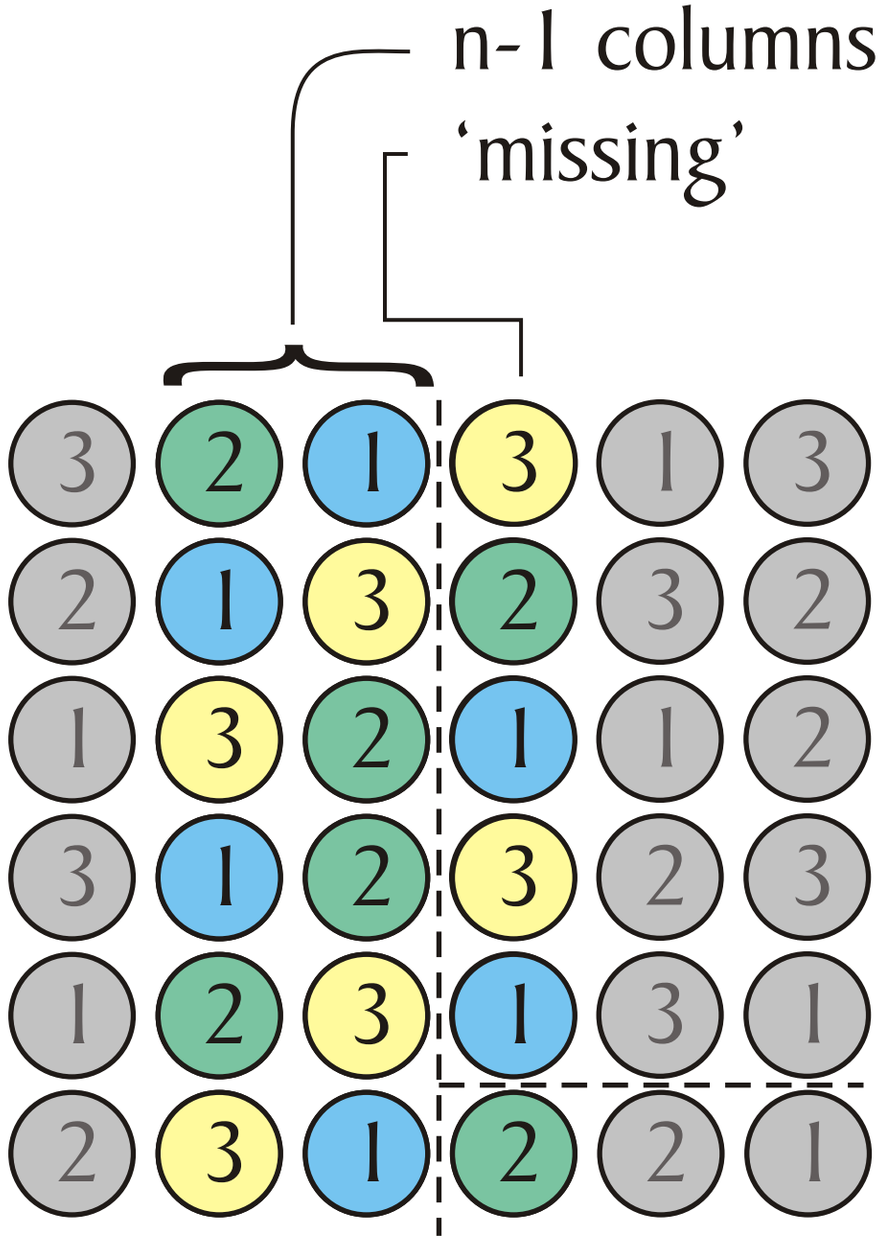
All Permutations

Circulant Property



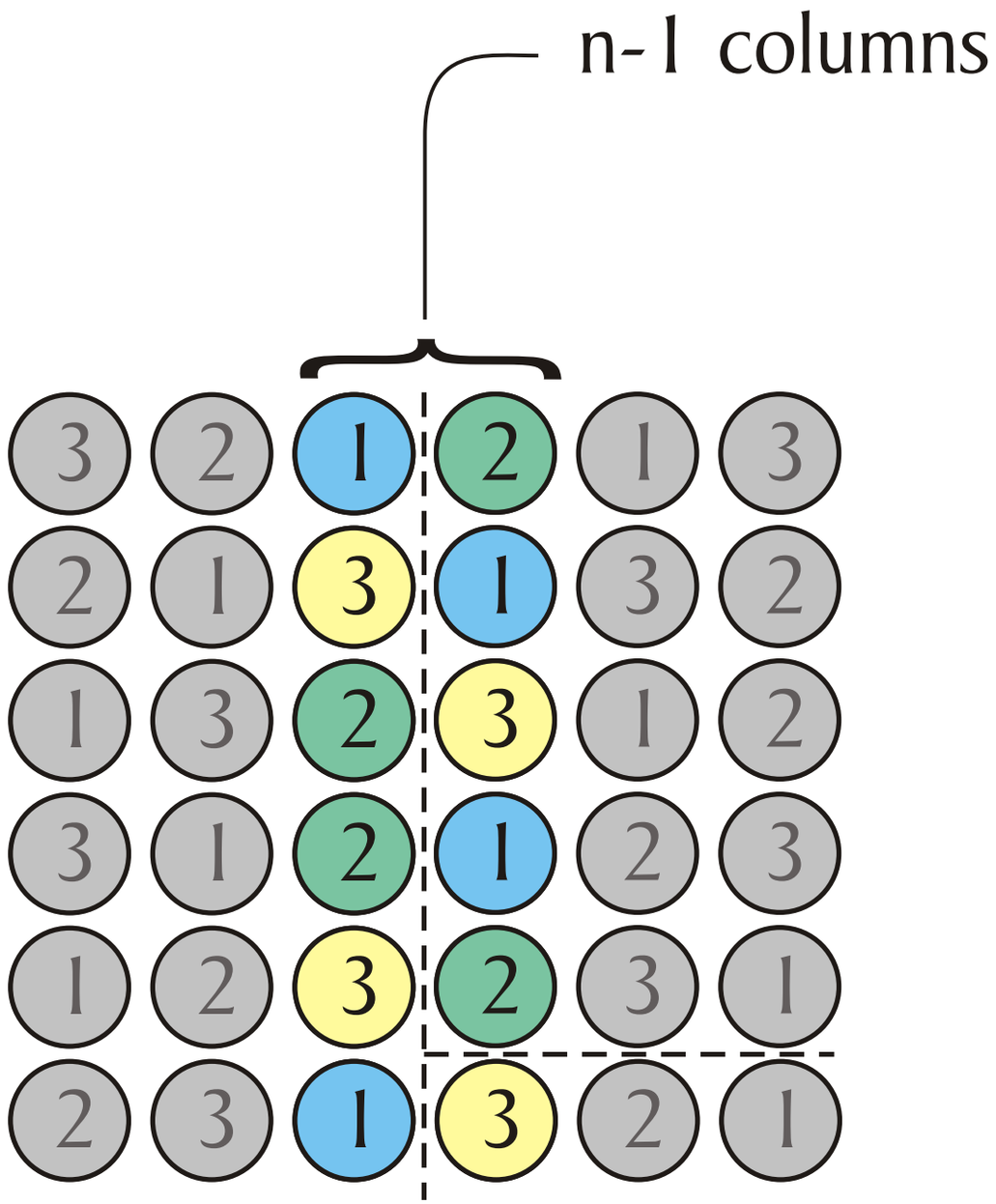
Offset 1

Circulant Property



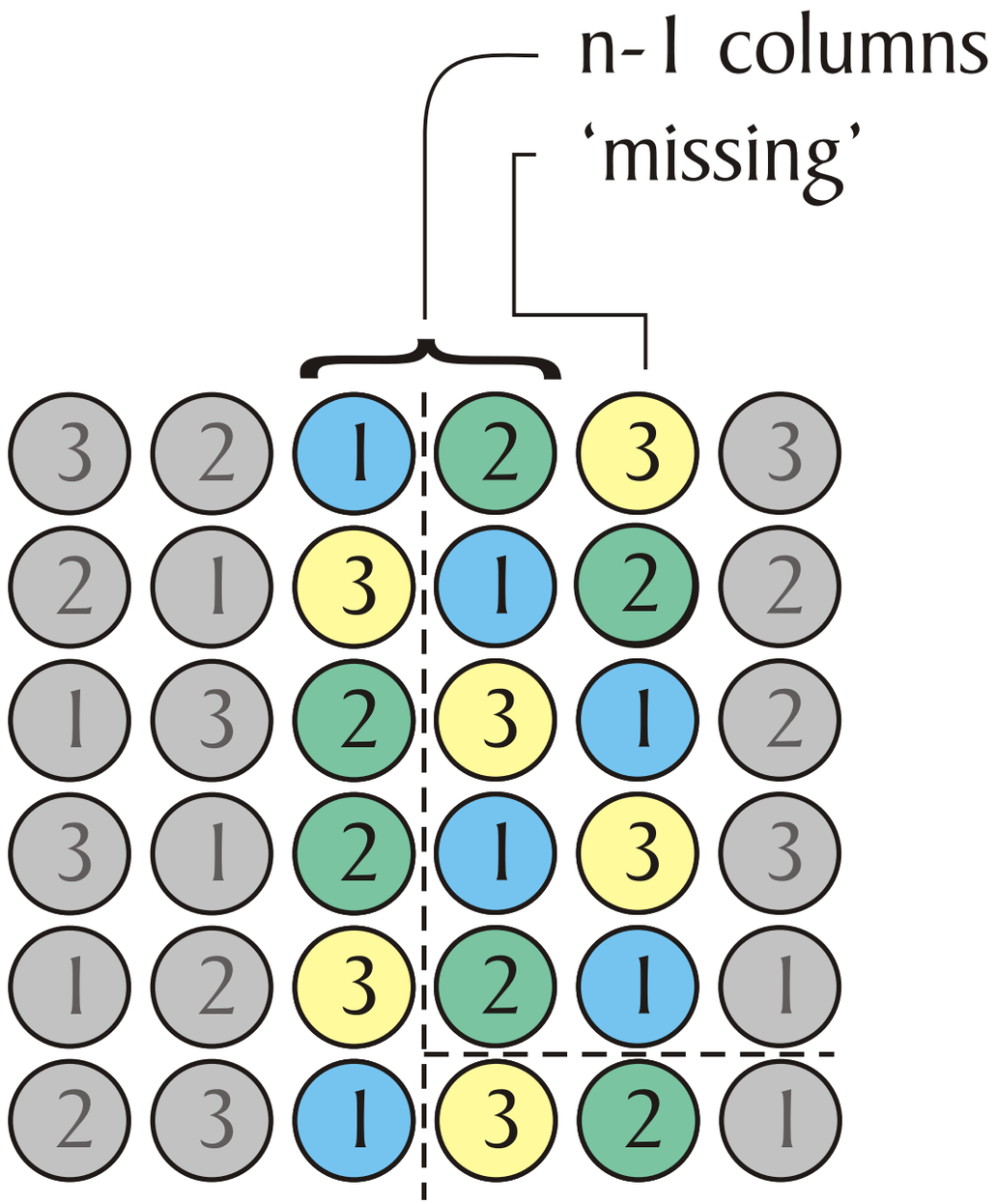
All Permutations

Circulant Property



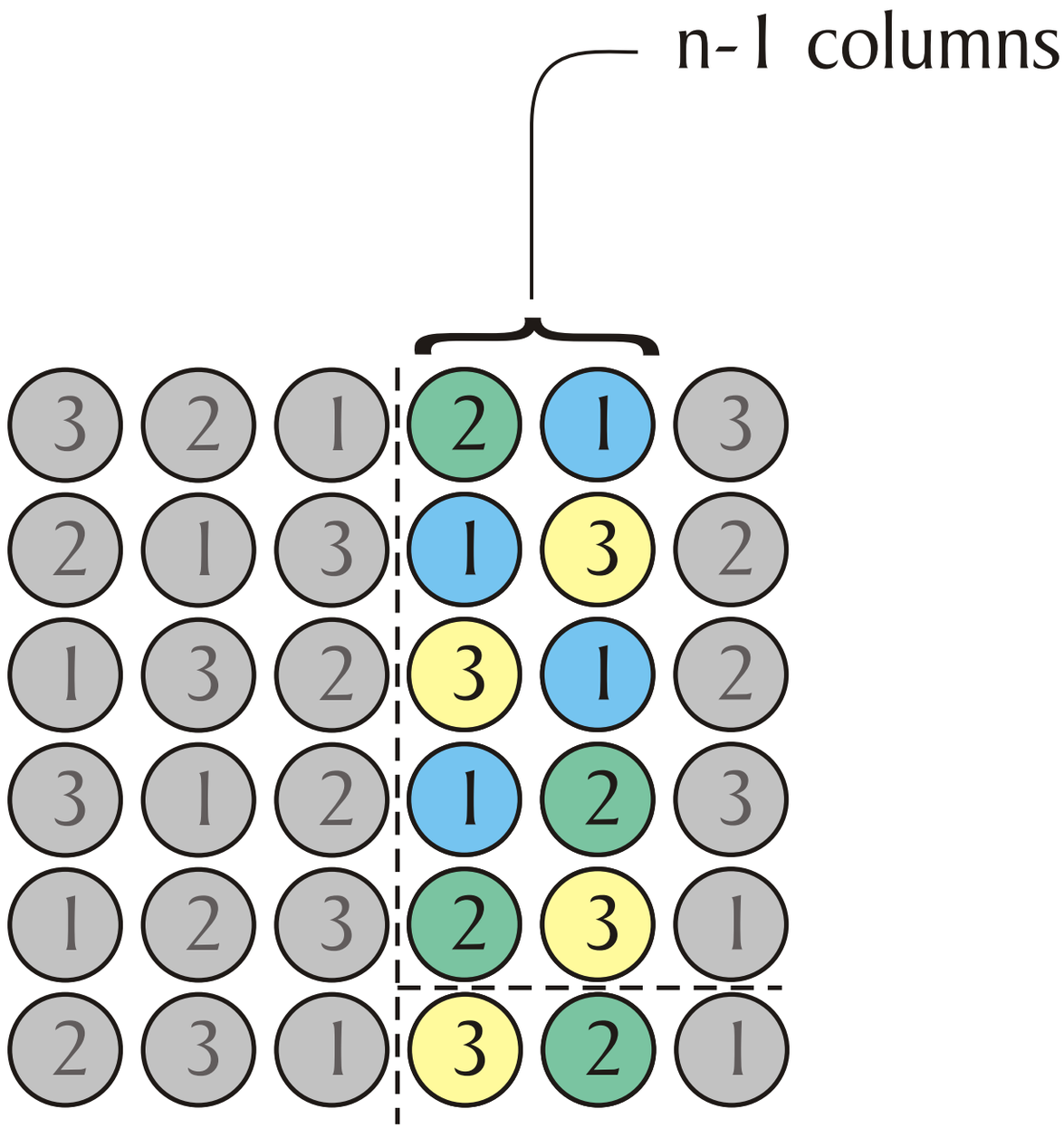
Offset 2

Circulant Property



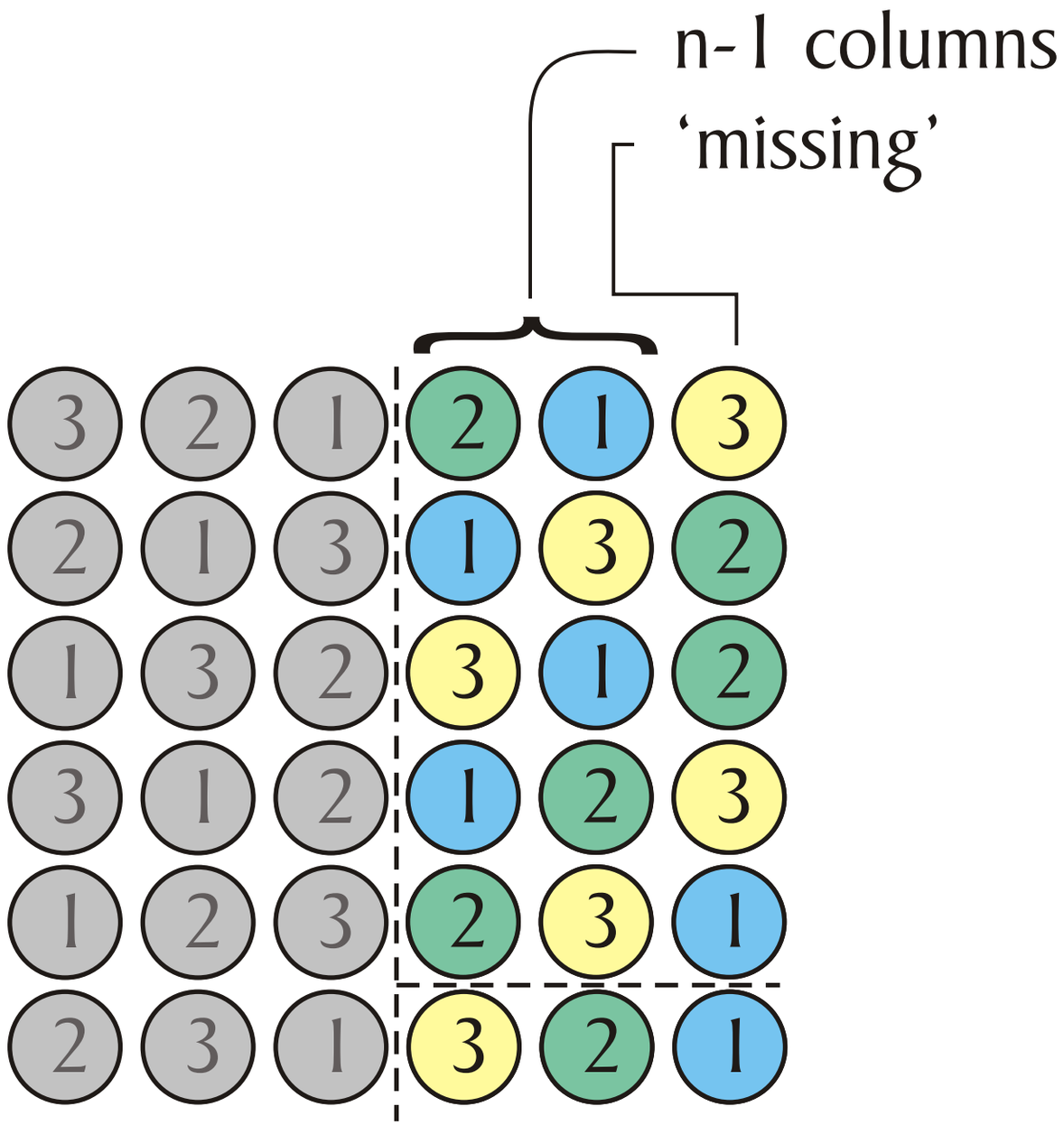
All Permutations

Circulant Property



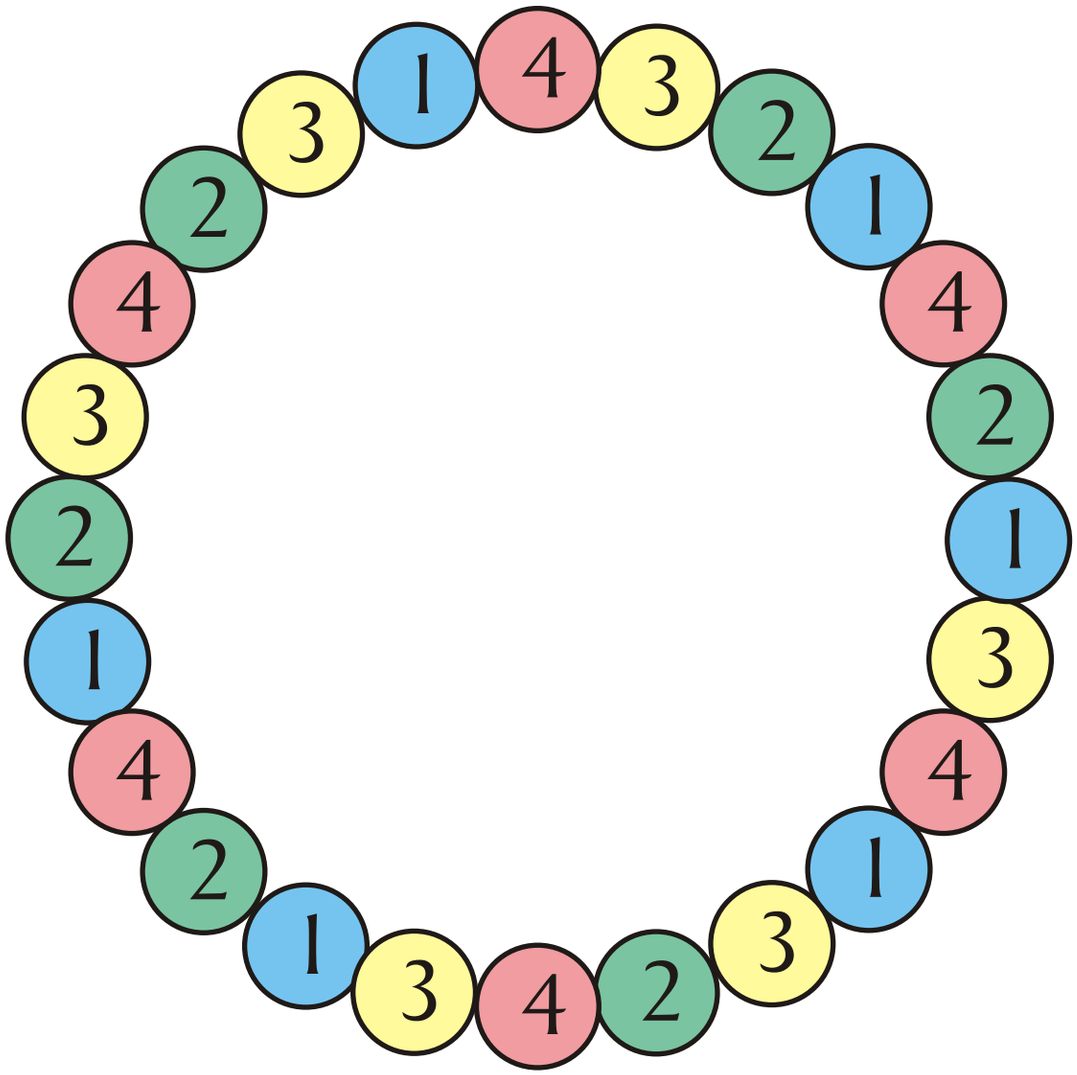
Offset 3

Circulant Property

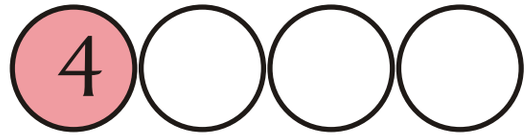


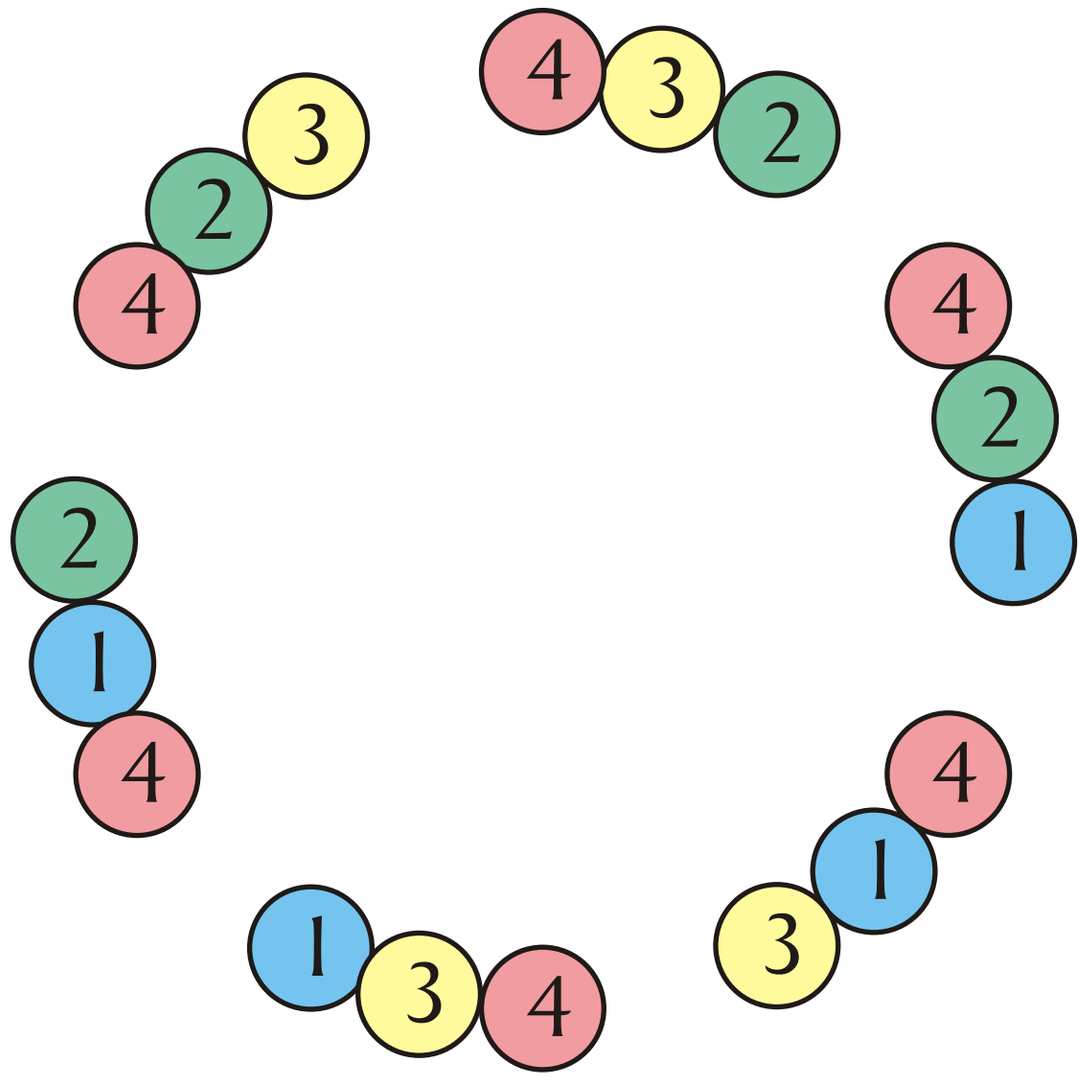
All Permutations

Construction

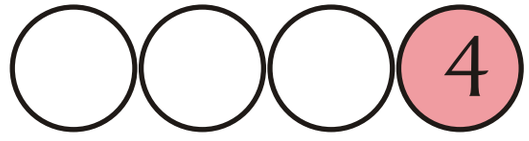


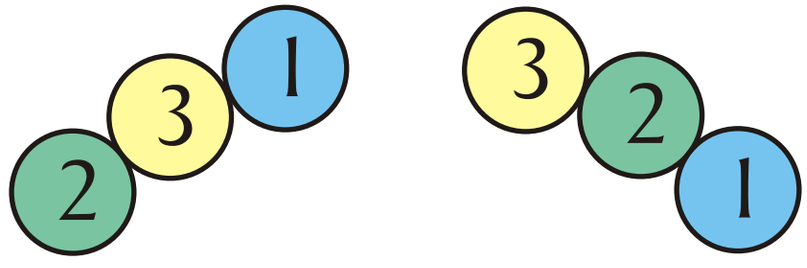
Construction

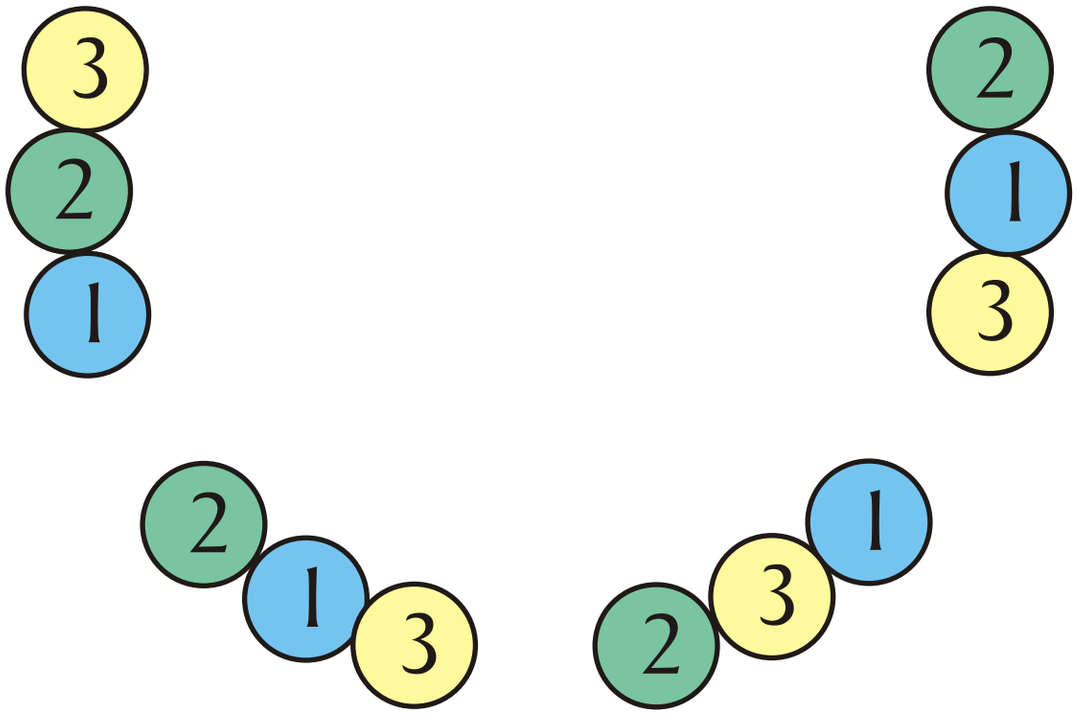
Substrings starting at 0 mod 4 contain permutations 



Construction

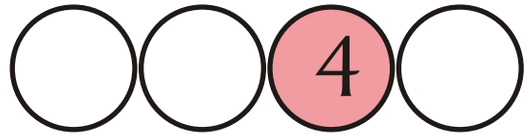
Substrings starting at 1 mod 4 contain permutations 

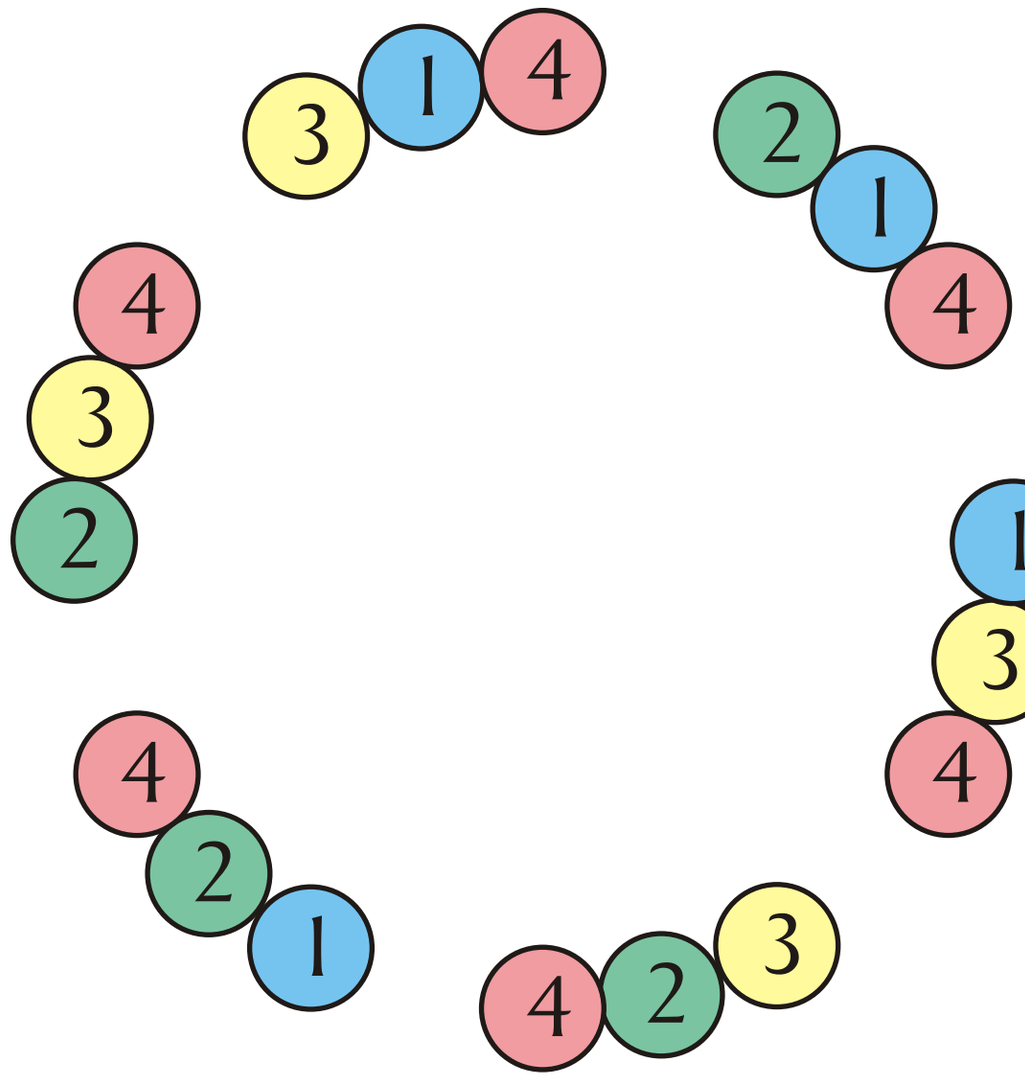




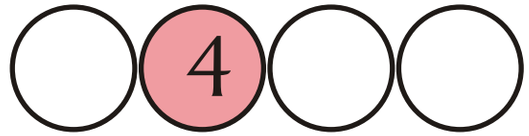
Construction

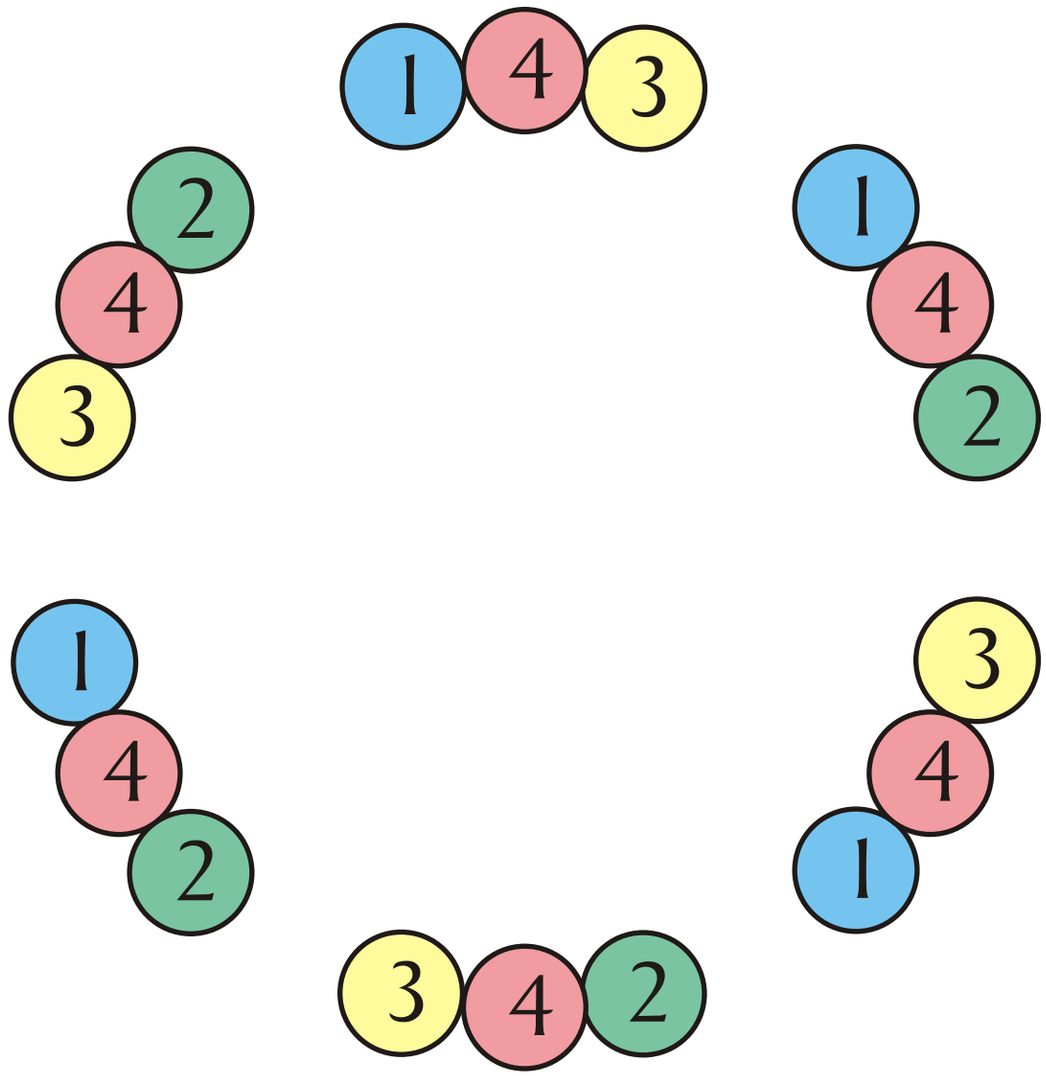
Substrings starting at 2 mod 4 contain permutations





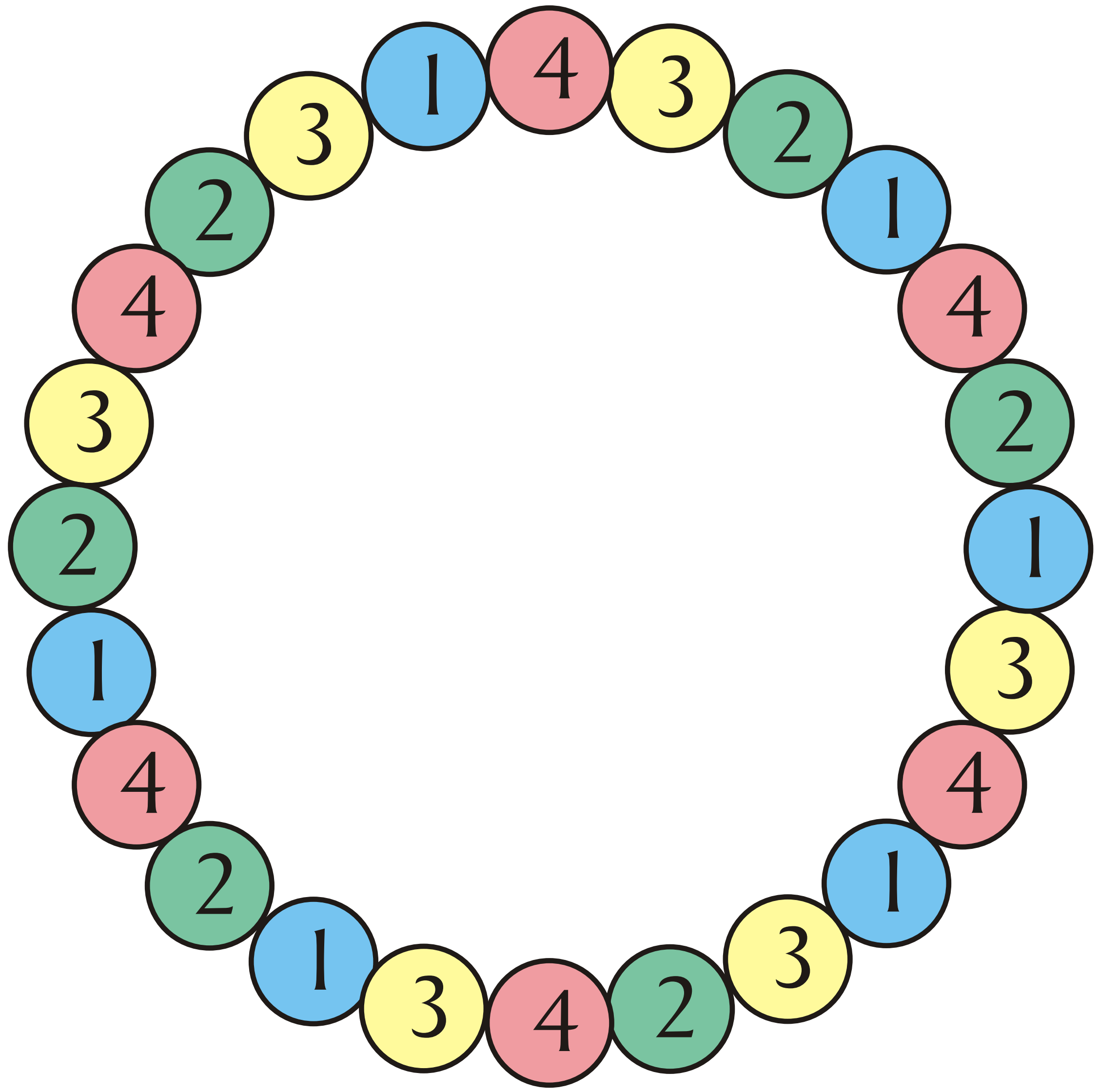
Construction

Substrings starting at 3 mod 4 contain permutations 



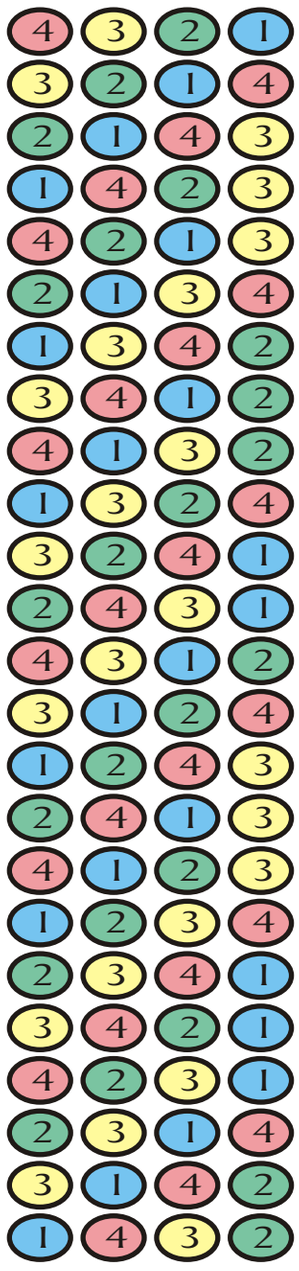
Construction

Collectively all permutations



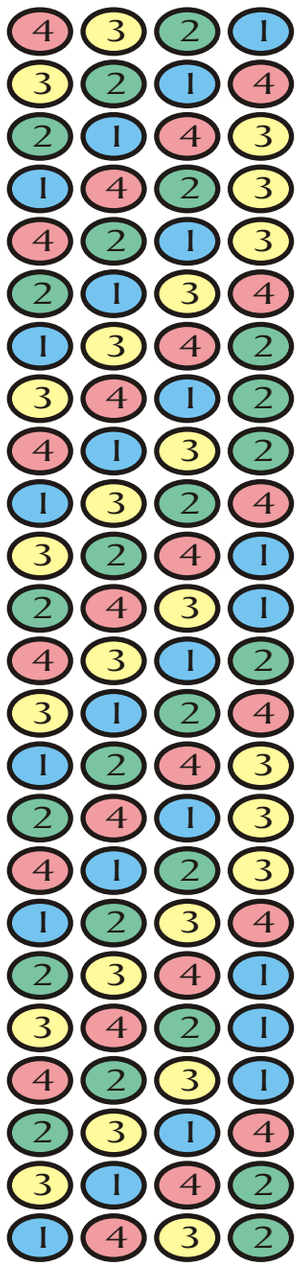
Algorithm for Gray Code

Use construction to iteratively generate permutations in loopless time with no extra memory



Algorithm for Gray Code

Use construction to iteratively generate permutations in loopless time with no extra memory

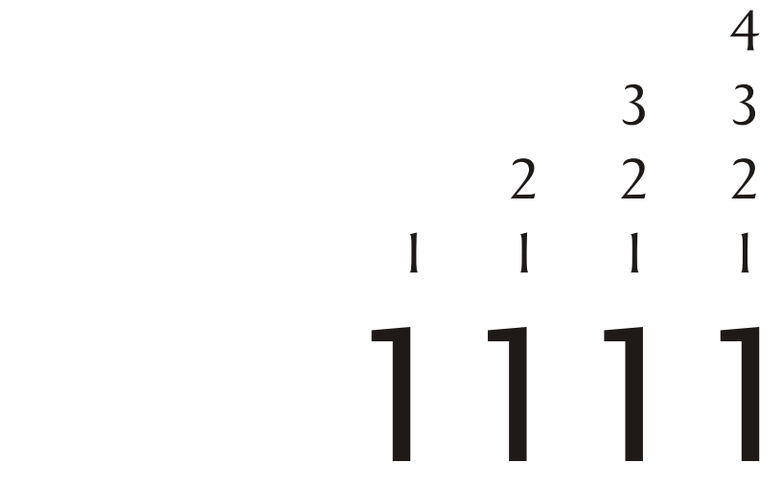


Modify loopless algorithm to count to n! Knuth

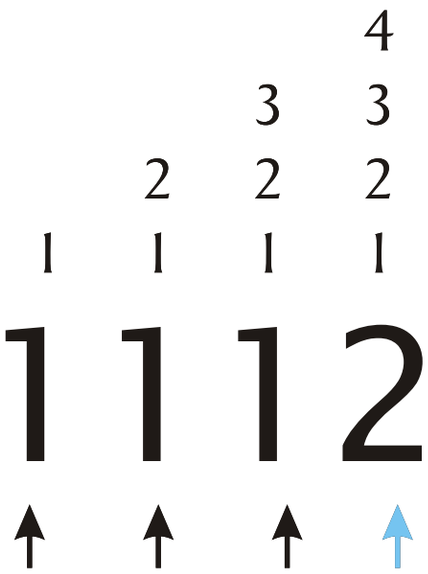
Counting Factorials

1111

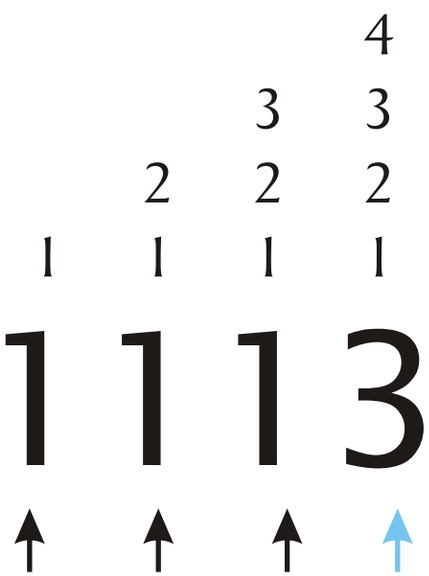
Counting Factorials



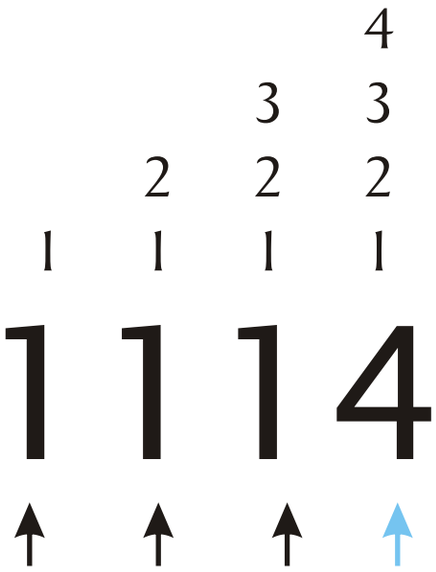
Counting Factorials



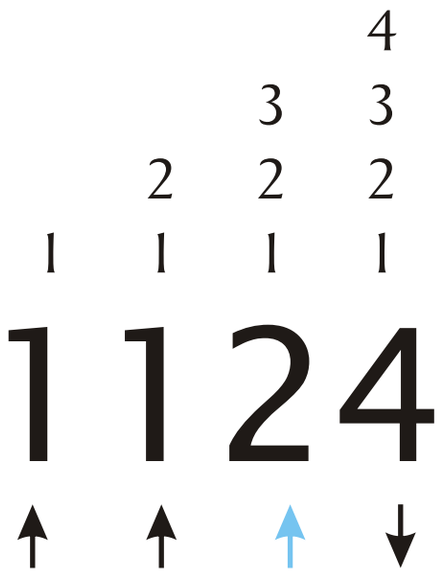
Counting Factorials



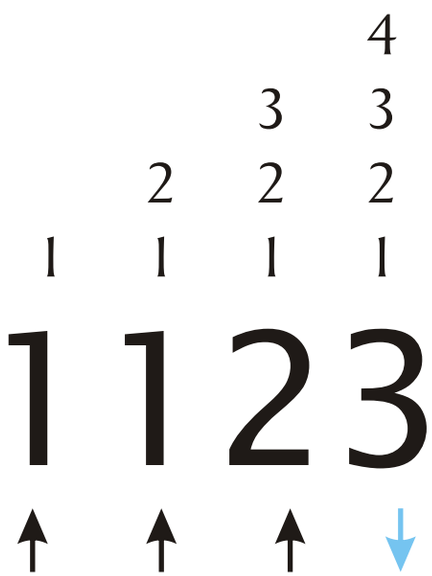
Counting Factorials



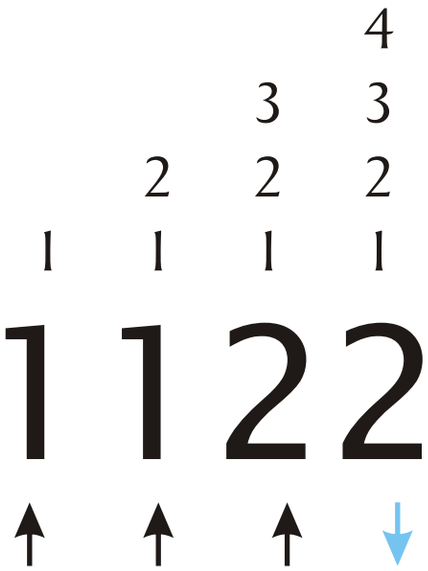
Counting Factorials



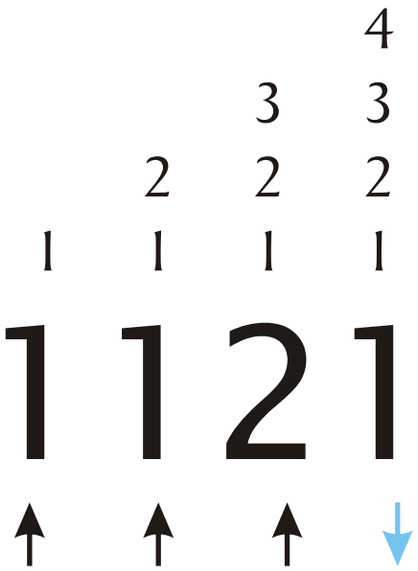
Counting Factorials



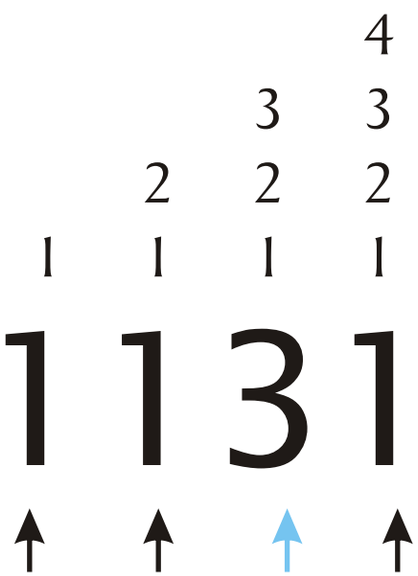
Counting Factorials



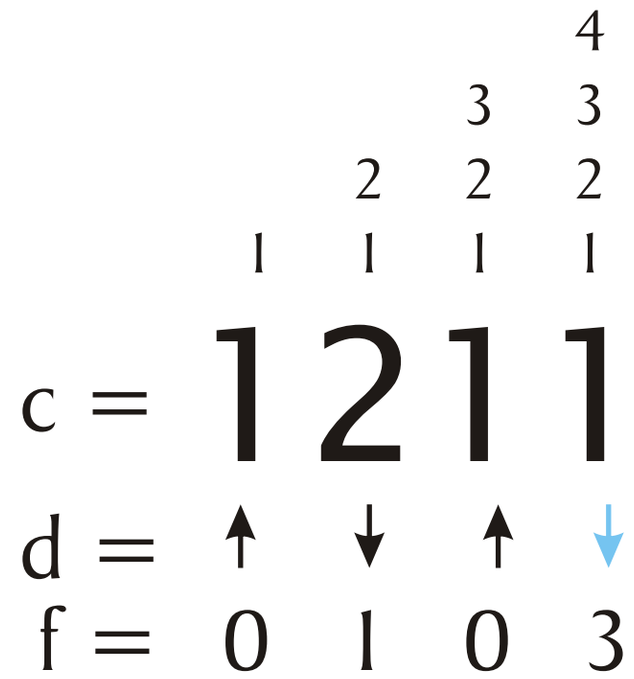
Counting Factorials



Counting Factorials



Counting Factorials



Array c where 1 <= c(i) <= i

Array d for directions

Array f focus pointers for which c(j) to change next

Counting Factorials

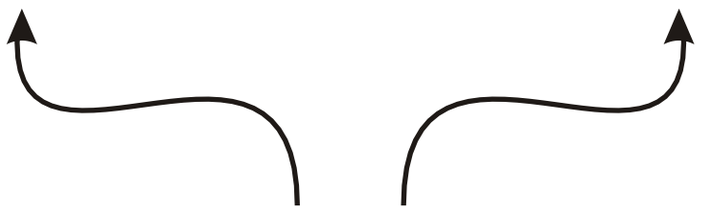
```
loop
     j := f(n)
     f(n) := n-1
     if j = 0
          return
     end
     c(j) := c(j) + d(j)
     if c(j) = 1 OR c(j) = j+1
          d(j) := -d(j)
          f(j+1) := f(j)
          f(j) := j-1
     end
end
```

Counting Factorials

```
loop
    j := f(n)
    f(n) := n-1
    if j = 0
        return
    end

    if mod(n-j,2) = 1 XOR (c(j)-d(j) <= 1 OR c(j)-d(j) >= j+1)
        rotate(perm, n-1)
    end
    if mod(n-j,2) = 1 XOR (c(j)-d(j) <= 1 OR c(j)-d(j) >= j+1)
        rotate(perm, n)
    end

    c(j) := c(j) + d(j)
    if c(j) = 1 OR c(j) = j+1
        d(j) := -d(j)
        f(j+1) := f(j)
        f(j) := j-1
    end
end
```

Counting Factorials

```
loop
    j := f(n)
    f(n) := n-1
    if j = 0
        return
    end

    if mod(n-j,2) = 1 XOR (c(j)-d(j) <= 1 OR c(j)-d(j) >= j+1)
        rotate(perm, n-1)
    end
        rotate(perm, n)
    end

    c(j) := c(j) + d(j)
    if c(j) = 1 OR c(j) = j+1
        d(j) := -d(j)
        f(j+1) := f(j)
        f(j) := j-1
    end
end
```



is the active element small and going down or large and going up?

Counting Factorials

```
loop
    j := f(n)
    f(n) := n-1
    if j = 0
        return
    end

    if mod(n-j,2) = 1 XOR (c(j)-d(j) <= 1 OR c(j)-d(j) >= j+1)
        rotate(perm, n-1)
    end
        rotate(perm, n)
    end

    c(j) := c(j) + d(j)
    if c(j) = 1 OR c(j) = j+1
        d(j) := -d(j)
        f(j+1) := f(j)
        f(j) := j-1
    end
end
```

is the active position odd?

is the active element small and going down
or large and going up?

Counting Factorials

```
loop
    j := f(n)
    f(n) := n-1
    if j = 0
        return
    end

    if mod(n-j,2) = 1 XOR (c(j)-d(j) <= 1 OR c(j)-d(j) >= j+1)
        rotate(perm, n-1)
    end
        rotate(perm, n)
    end

    c(j) := c(j) + d(j)
    if c(j) = 1 OR c(j) = j+1
        d(j) := -d(j)
        f(j+1) := f(j)
        f(j) := j-1
    end
end
```

is the active position odd?

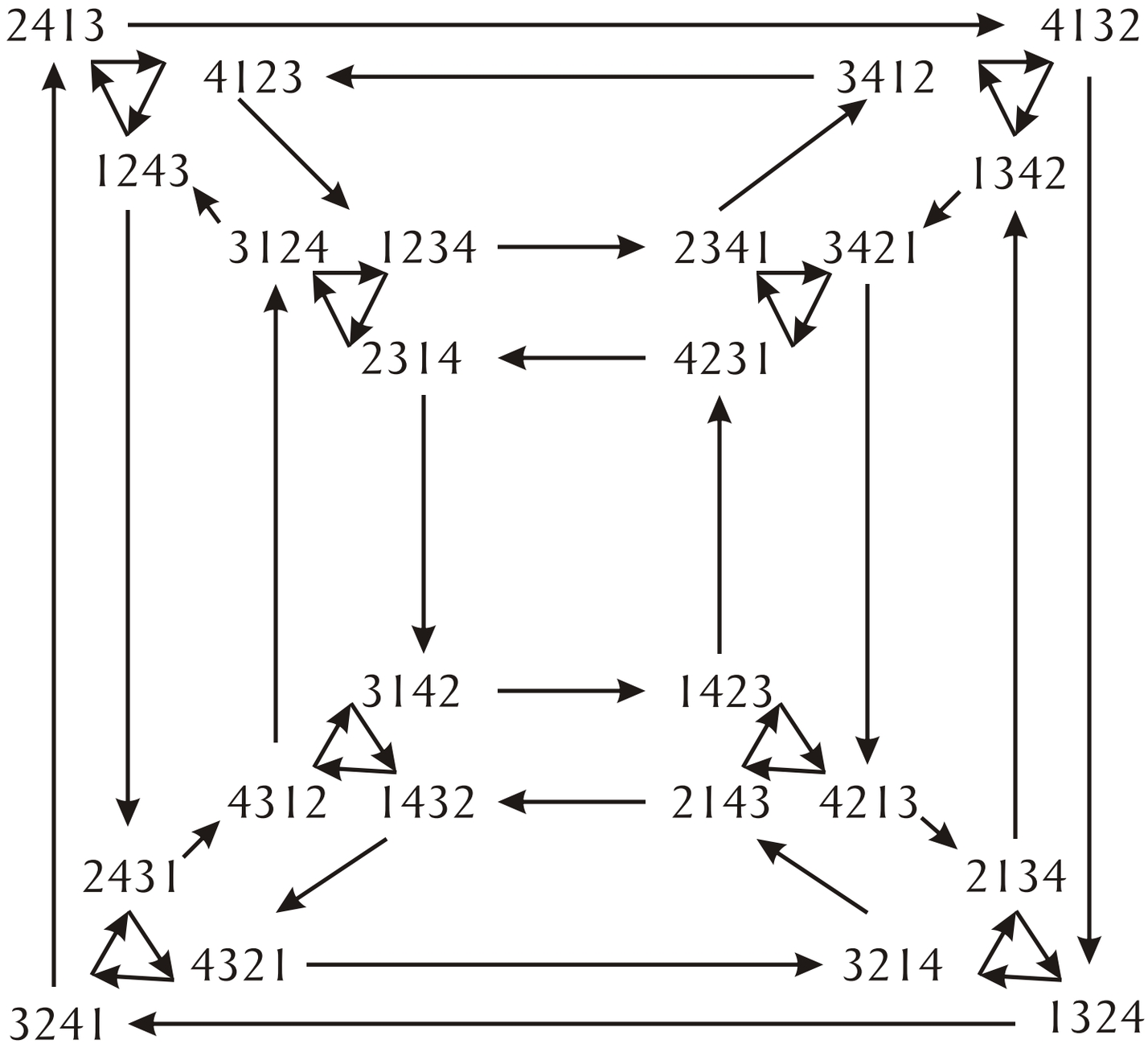
is the active element small and going down
or large and going up?

Condition can be simplified to `if m(j)=1` by maintaining binary array `m`

Additional Shorthand Universal Cycles

Additional Shorthand Universal Cycles

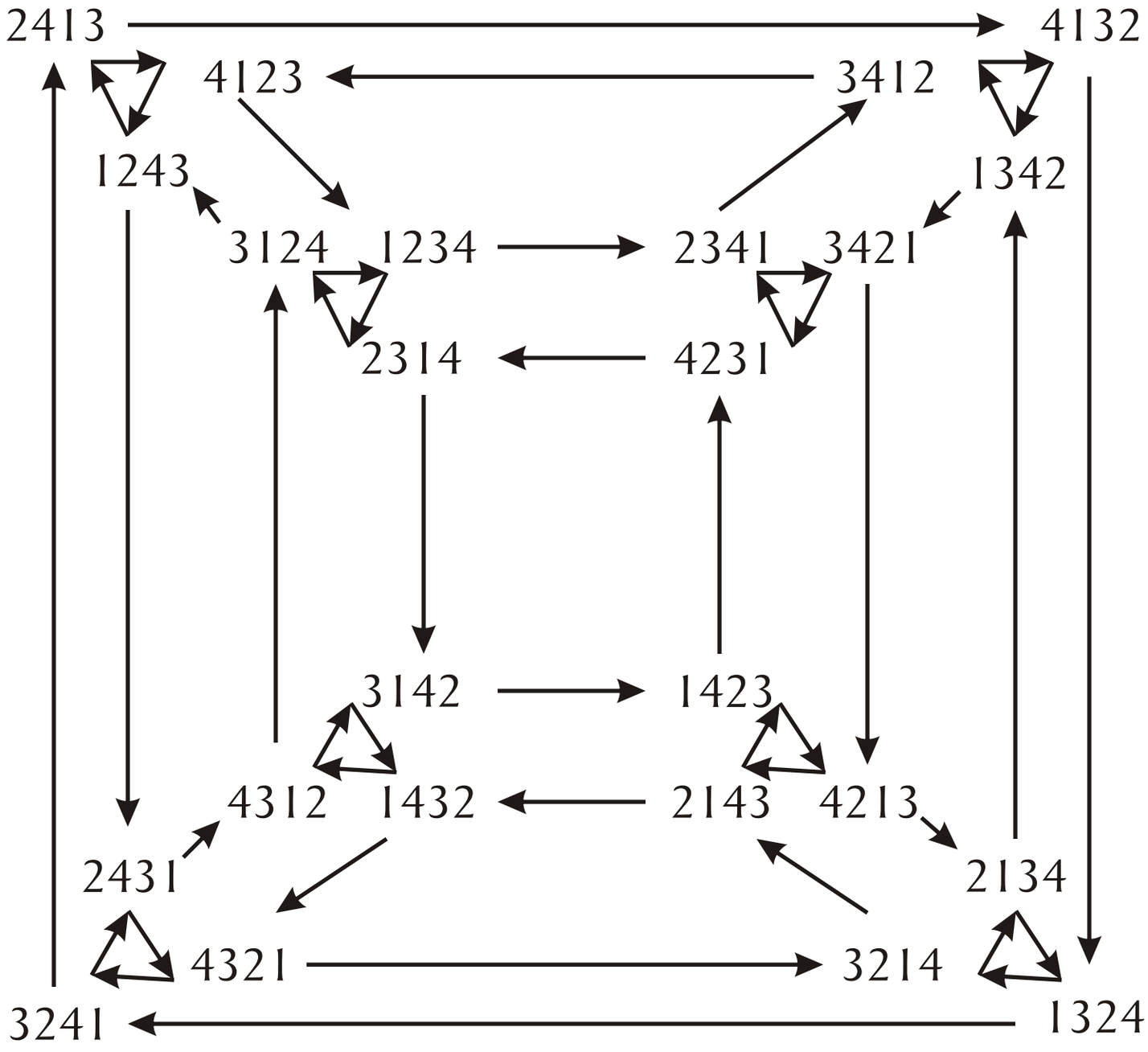
G =



Cayley Graph with rotation generators n-1 and n

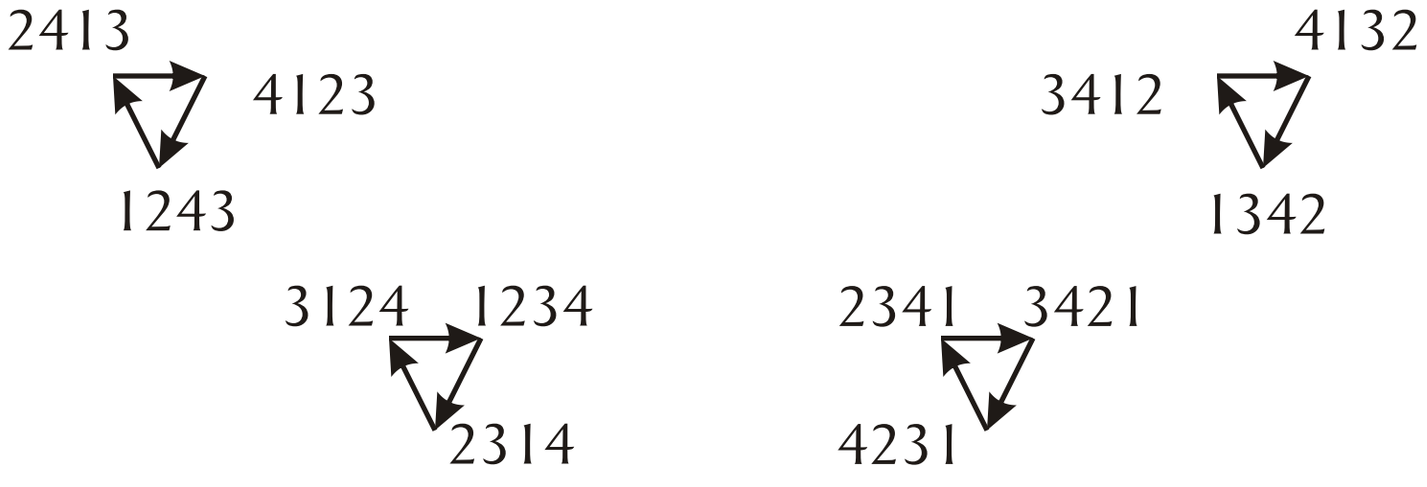
Additional Shorthand Universal Cycles

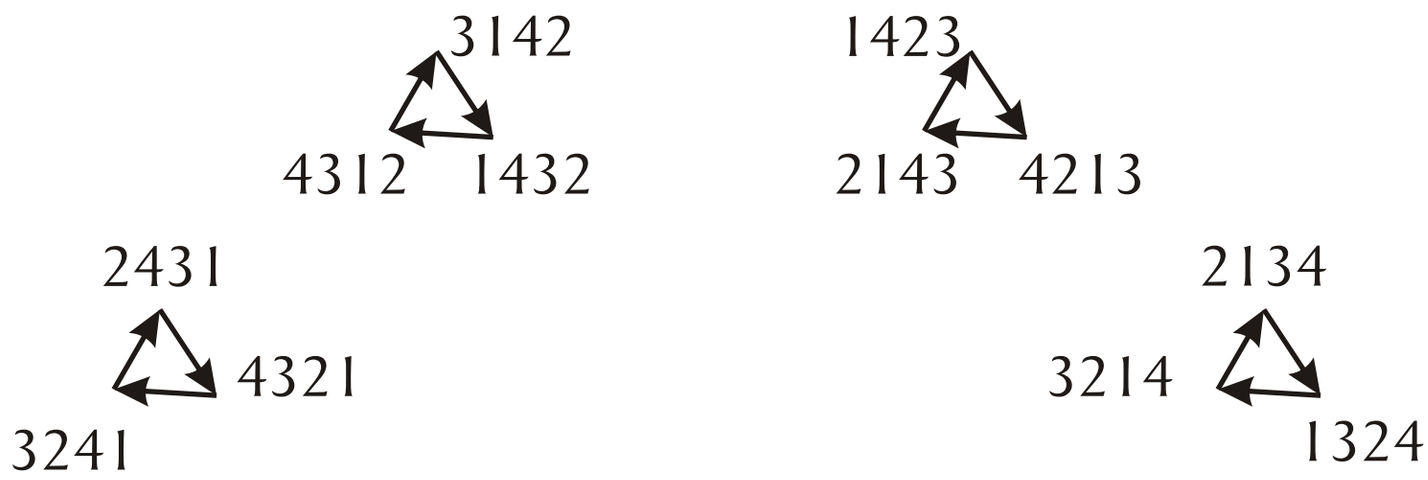
G =



Hamilton cycles are equivalent to Shorthand Universal Cycles

Additional Shorthand Universal Cycles





Shrink directed cycles of length n-1

Additional Shorthand Universal Cycles

124

134

123

234

143

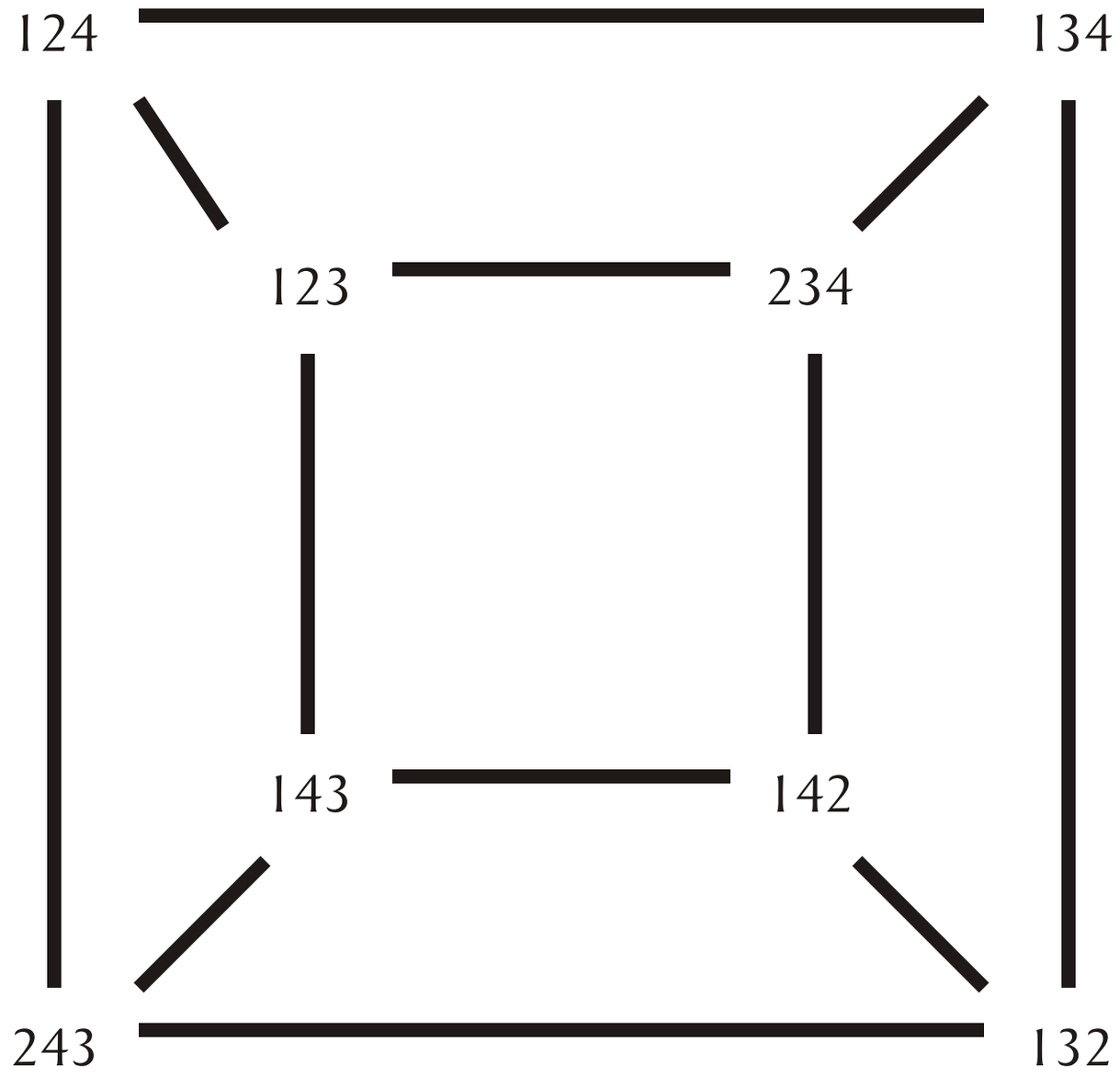
142

243

132

Vertices are necklace classes on n-1 symbols

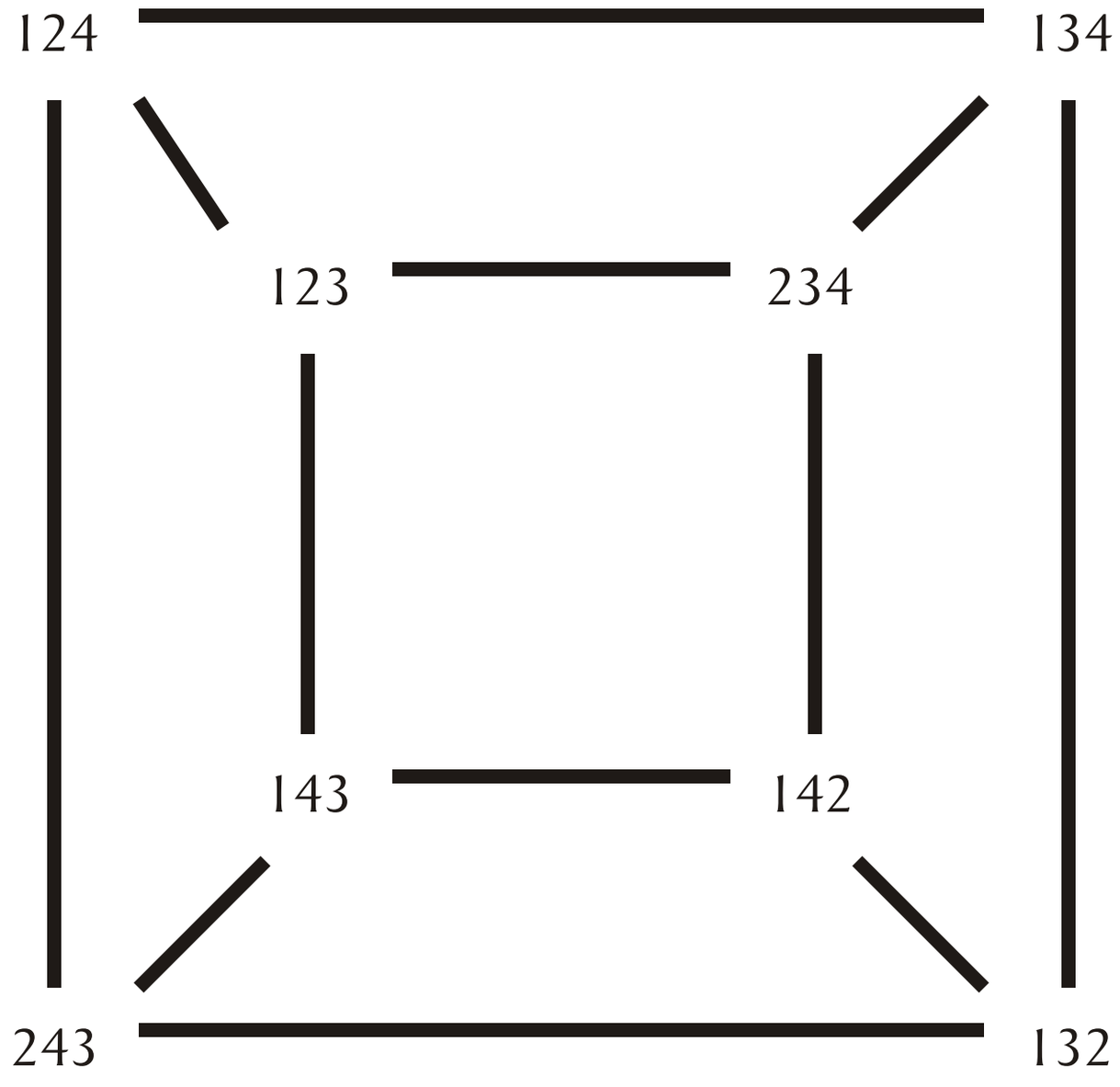
Additional Shorthand Universal Cycles



Connect necklaces

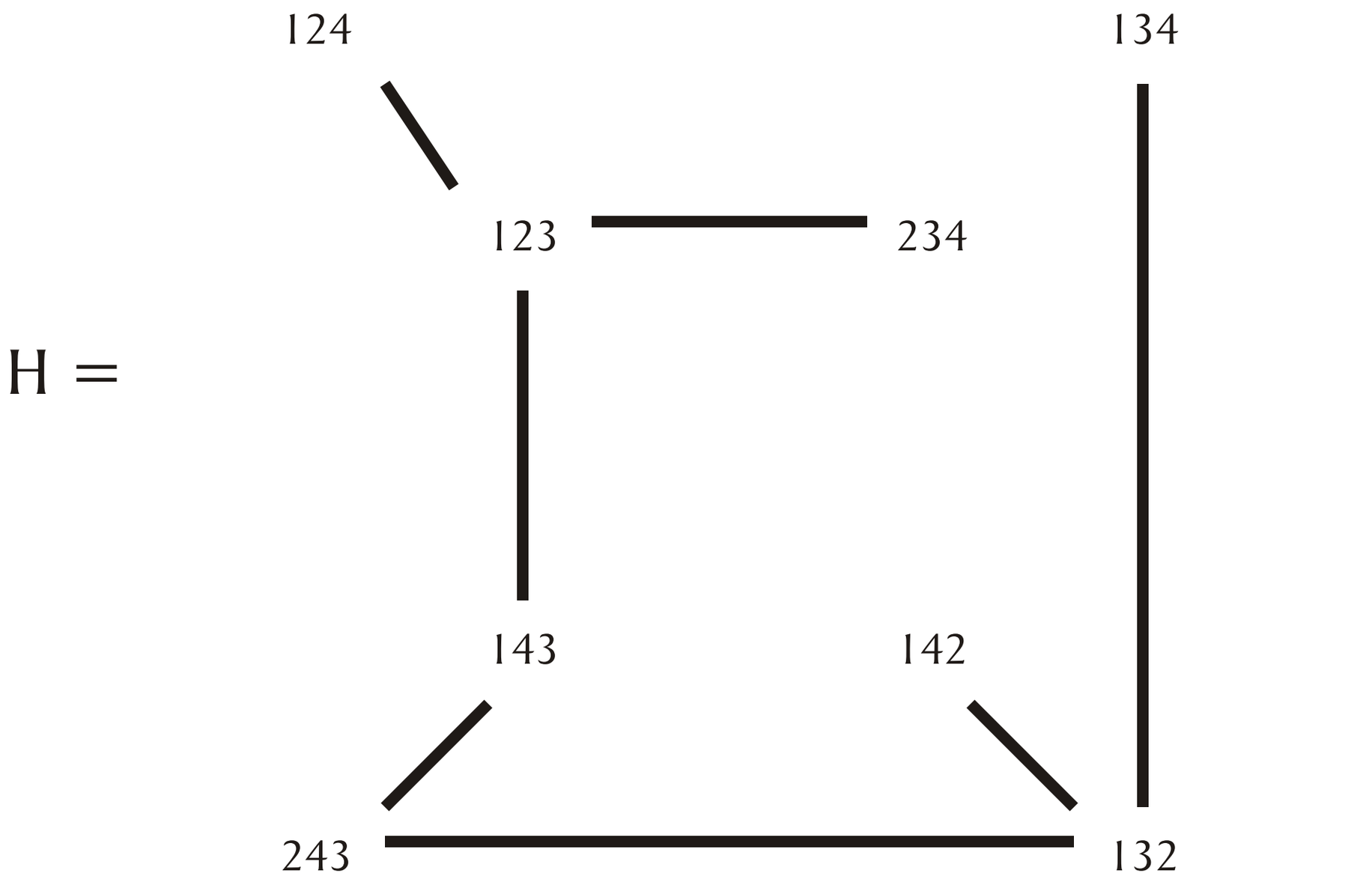
Additional Shorthand Universal Cycles

H =



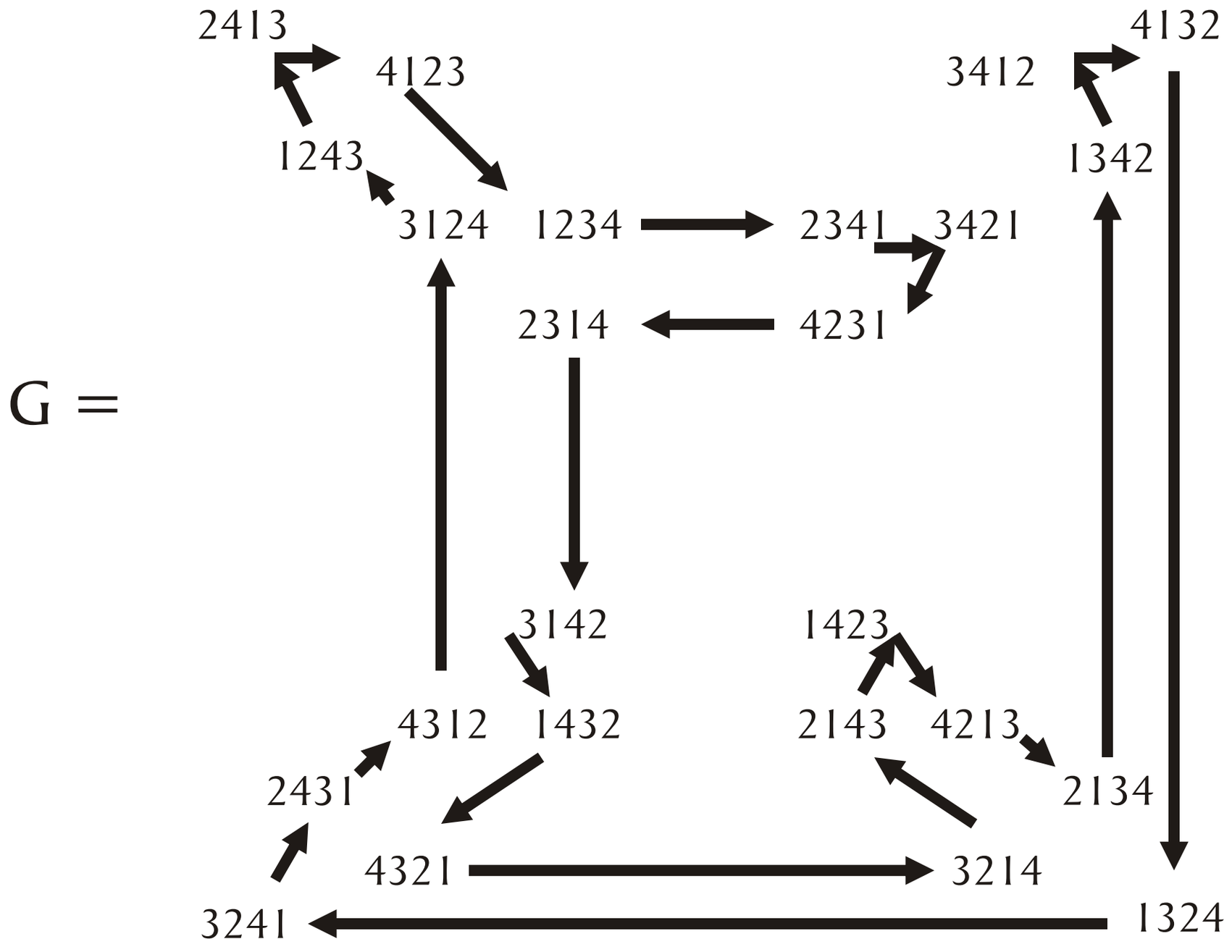
Spanning Trees in H become Hamilton cycles in G

Additional Shorthand Universal Cycles



Spanning Trees in H become Hamilton cycles in G

Additional Shorthand Universal Cycles

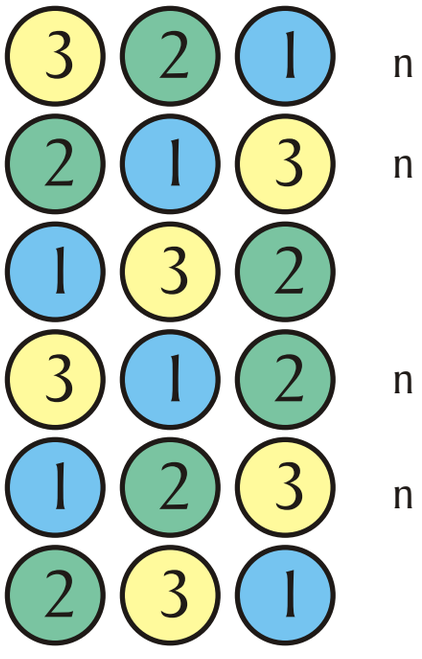


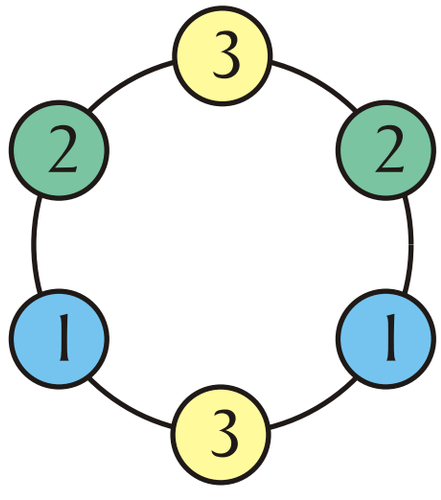
Spanning Trees in H become Hamilton cycles in G

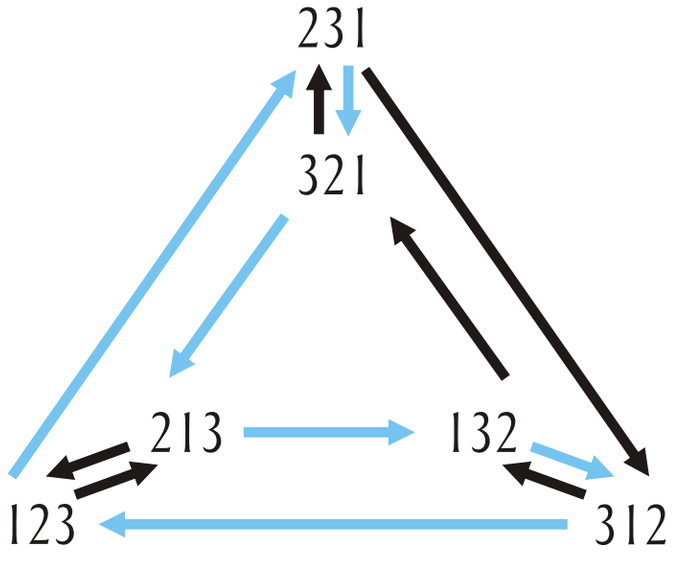
Summary

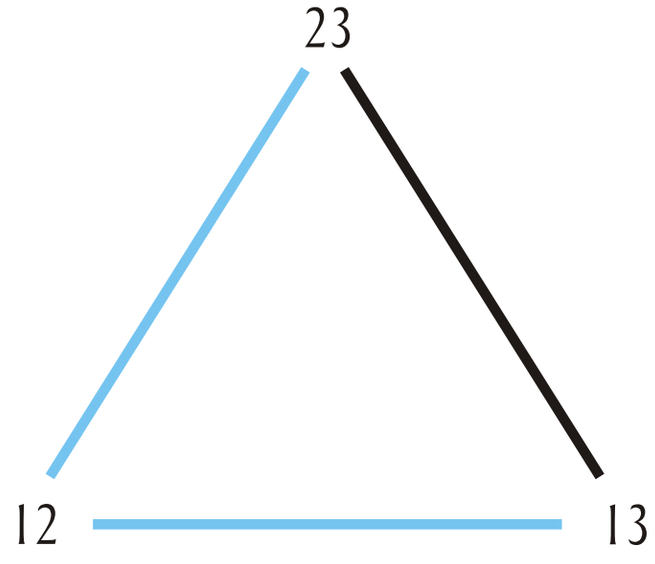
The following are equivalent

- Gray codes for Permutations by rotations of length n-1 and n
- Shorthand Universal Cycles for Permutations
- Hamilton Cycles in Cayley Graph with rotation generators n-1 and n
- (Spanning Trees in n-1 Necklace Graph)





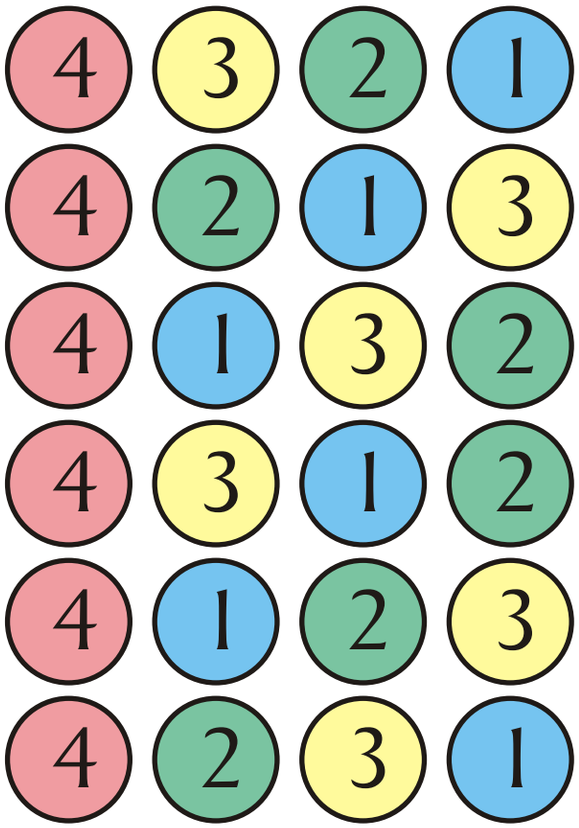


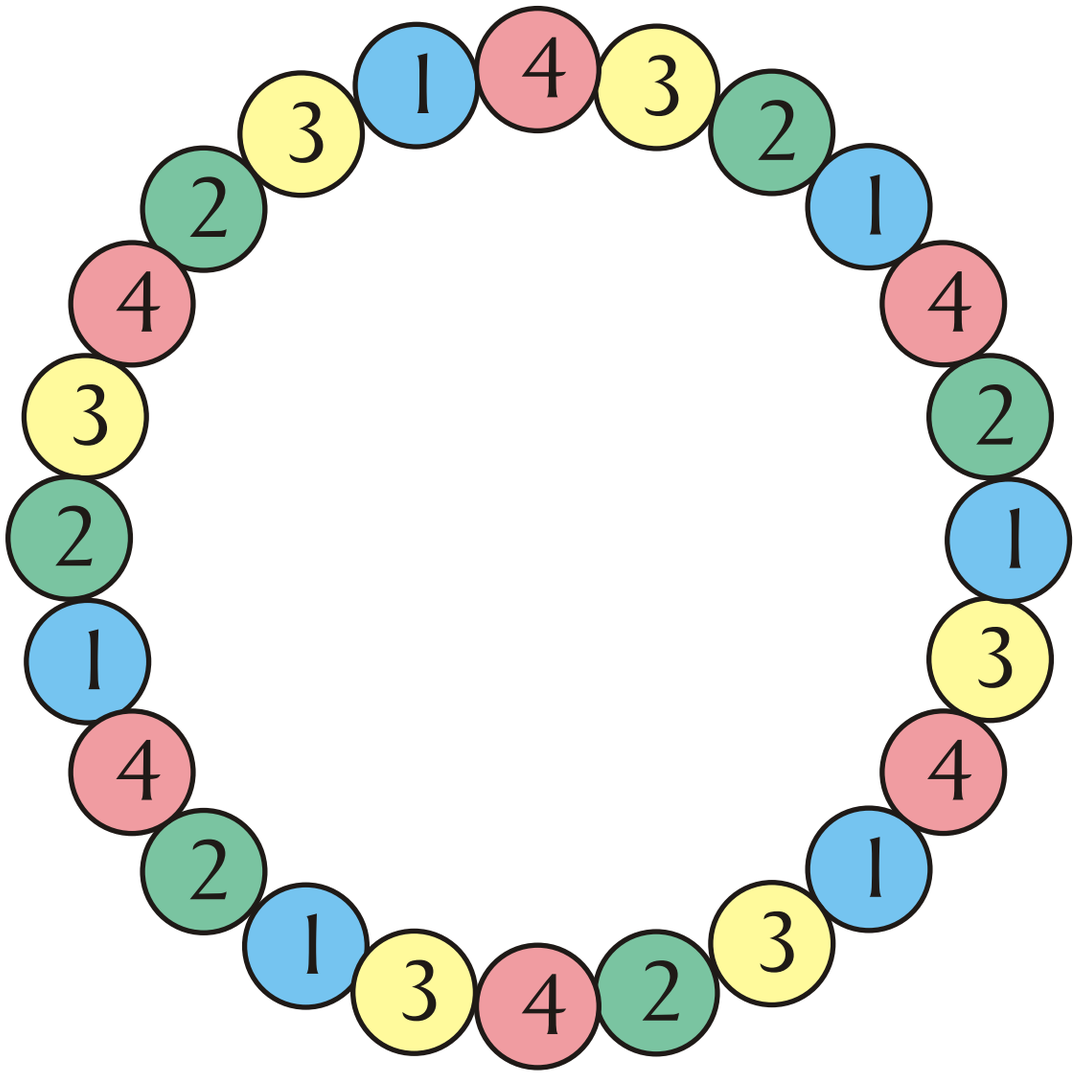


Summary

Shorthand Universal Cycles can be constructed recursively

- Predictable symbol method
- Circulant Property

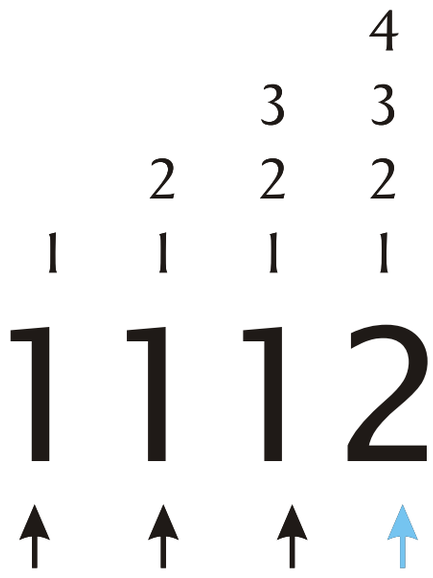




Summary

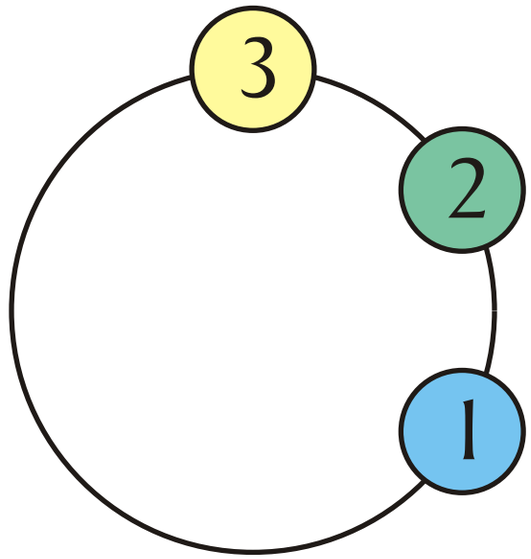
Gray codes can be generated in loopless time

Modify loopless mixed-radix algorithm



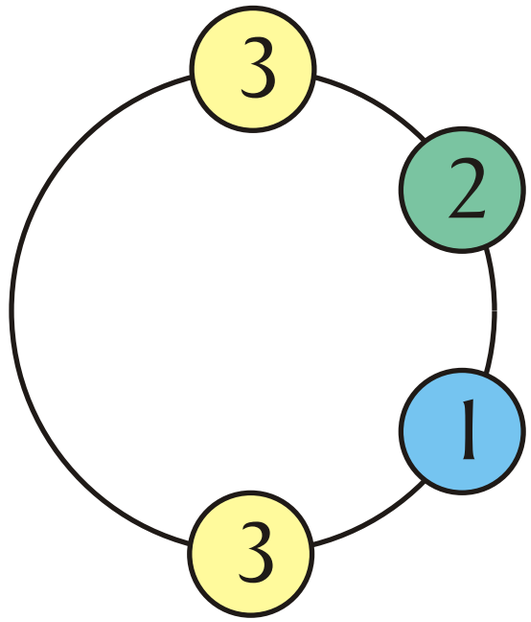
Relation to Previous Research

Universal Cycles for Permutations do not exist Chung Diaconis Graham



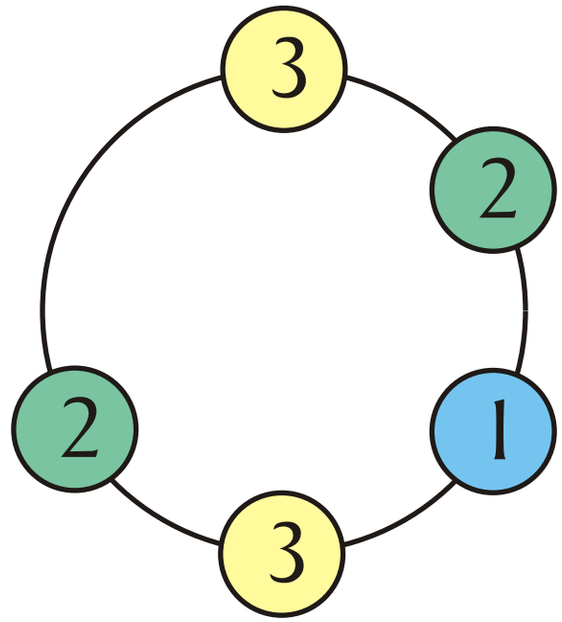
Relation to Previous Research

Universal Cycles for Permutations do not exist Chung Diaconis Graham



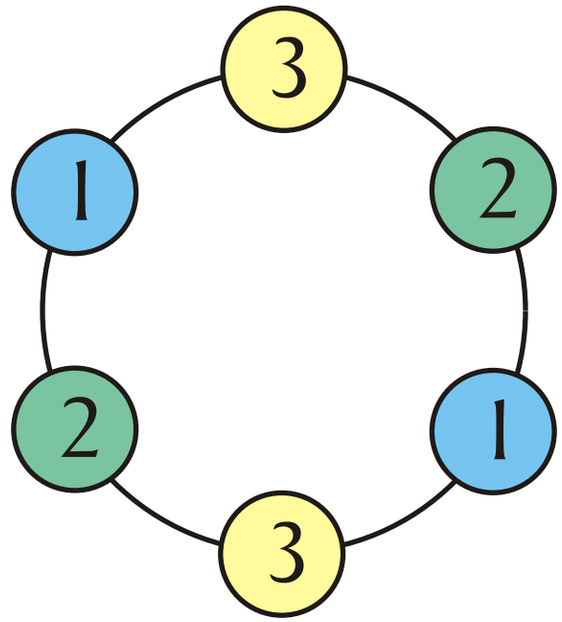
Relation to Previous Research

Universal Cycles for Permutations do not exist Chung Diaconis Graham



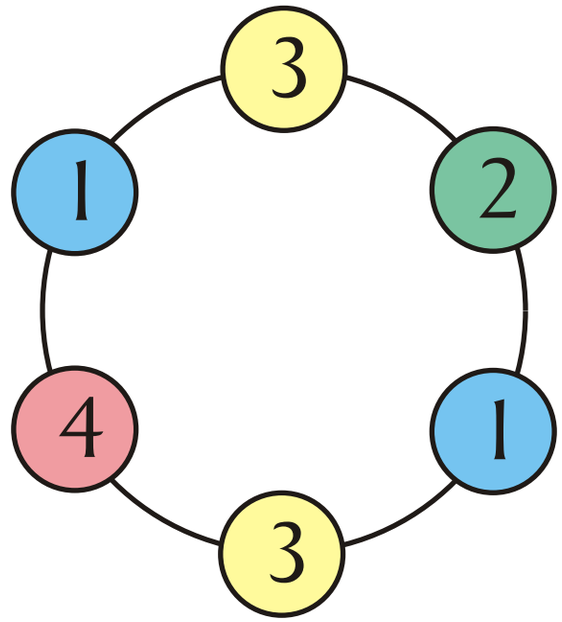
Relation to Previous Research

Universal Cycles for Permutations do not exist Chung Diaconis Graham



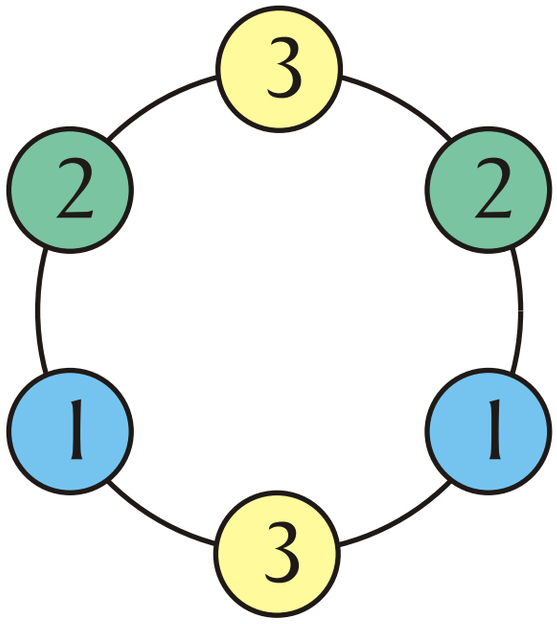
Relation to Previous Research

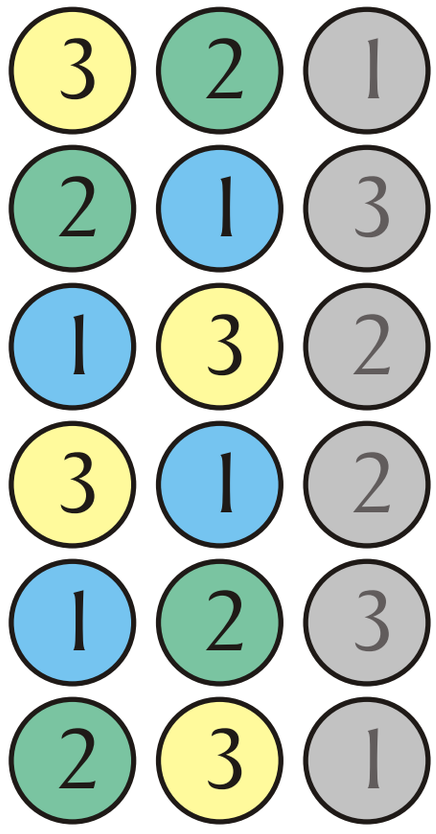
Order-isomorphic exist with n+1 symbols Johnson



Relation to Previous Research

Equivalent to Universal Cycles for (n-1)-Permutations Jackson Knuth





Additional Results and Open Problems

Extend to Permutations of a Multi-set

Extend to k-Permutations

Circulant Property

Rotations of length n and one of 2,3,...,n-2? Cheng

Thanks!