Combinatorial Objects

Permutations

k-element Subsets

Gray Codes

Generating, Ranking, Unranking

Numbers 1, 2, ..., N
can be perceived as just labels or indexes of some other items/objects x1, x2, ..., xN.

All ideas discussed here apply to the permutations of {1, 2, ..., N}
and to the permutations of {x1, x2, ..., xN} in the same way.

Example:

Permutations of set of size 3

set {1, 2, 3}

{1, 2, 3}
{1, 3, 2}

{2, 1, 3}
{2, 3, 1}

{3, 1, 2}
{3, 2, 1}

index { 1, 2, 3 }
set { Ann, Bob, Don }

{Ann, Bob, Don}
{Ann, Don, Bob}

{Bob, Ann, Don}
{Bob, Don, Ann }

{Don, Ann, Bob}
{Don, Bob, Ann}

Studying permutations of {1,2,3,4,5} -- list of permutations with their ranks

0 (1 2 3 4 5)	24 (2 1 3 4 5)	48 (3 1 2 4 5)	72 (4 1 2 3 5)	96 (5 1 2 3 4)
1 (1 2 3 5 4)	25 (2 1 3 5 4)	49 (3 1 2 5 4)	73 (4 1 2 5 3)	97 (5 1 2 4 3)
2 (1 2 4 3 5)	26 (2 1 4 3 5)	50 (3 1 4 2 5)	74 (4 1 3 2 5)	98 (5 1 3 2 4)
3 (1 2 4 5 3)	27 (2 1 4 5 3)	51 (3 1 4 5 2)	75 (4 1 3 5 2)	99 (5 1 3 4 2)
4 (1 2 5 3 4)	28 (2 1 5 3 4)	52 (3 1 5 2 4)	76 (4 1 5 2 3)	100 (5 1 4 2 3)
5 (1 2 5 4 3)	29 (2 1 5 4 3)	53 (3 1 5 4 2)	77 (4 1 5 3 2)	101 (5 1 4 3 2)
6 (1 3 2 4 5)	30 (2 3 1 4 5)	54 (3 2 1 4 5)	78 (4 2 1 3 5)	102 (5 2 1 3 4)
7 (1 3 2 5 4)	31 (2 3 1 5 4)	55 (3 2 1 5 4)	79 (4 2 1 5 3)	103 (5 2 1 4 3)
8 (1 3 4 2 5)	32 (2 3 4 1 5)	56 (3 2 4 1 5)	80 (4 2 3 1 5)	104 (5 2 3 1 4)
9 (1 3 4 5 2)	33 (2 3 4 5 1)	57 (3 2 4 5 1)	81 (4 2 3 5 1)	105 (5 2 3 4 1)
10 (1 3 5 2 4)	34 (2 3 5 1 4)	58 (3 2 5 1 4)	82 (4 2 5 1 3)	106 (5 2 4 1 3)
11 (1 3 5 4 2)	35 (2 3 5 4 1)	59 (3 2 5 4 1)	83 (4 2 5 3 1)	107 (5 2 4 3 1)
12 (1 4 2 3 5)	36 (2 4 1 3 5)	60 (3 4 1 2 5)	84 (4 3 1 2 5)	108 (5 3 1 2 4)
13 (1 4 2 5 3)	37 (2 4 1 5 3)	61 (3 4 1 5 2)	85 (4 3 1 5 2)	109 (5 3 1 4 2)
14 (1 4 3 2 5)	38 (2 4 3 1 5)	62 (3 4 2 1 5)	86 (4 3 2 1 5)	110 (5 3 2 1 4)
15 (1 4 3 5 2)	39 (2 4 3 5 1)	63 (3 4 2 5 1)	87 (4 3 2 5 1)	111 (5 3 2 4 1)
16 (1 4 5 2 3)	40 (2 4 5 1 3)	64 (3 4 5 1 2)	88 (4 3 5 1 2)	112 (5 3 4 1 2)
17 (1 4 5 3 2)	41 (2 4 5 3 1)	65 (3 4 5 2 1)	89 (4 3 5 2 1)	113 (5 3 4 2 1)
18 (1 5 2 3 4)	42 (2 5 1 3 4)	66 (3 5 1 2 4)	90 (4 5 1 2 3)	114 (5 4 1 2 3)
19 (1 5 2 4 3)	43 (2 5 1 4 3)	67 (3 5 1 4 2)	91 (4 5 1 3 2)	115 (5 4 1 3 2)
20 (1 5 3 2 4)	44 (2 5 3 1 4)	68 (3 5 2 1 4)	92 (4 5 2 1 3)	116 (5 4 2 1 3)
21 (1 5 3 4 2)	45 (2 5 3 4 1)	69 (3 5 2 4 1)	93 (4 5 2 3 1)	117 (5 4 2 3 1)
22 (1 5 4 2 3)	46 (2 5 4 1 3)	70 (3 5 4 1 2)	94 (4 5 3 1 2)	118 (5 4 3 1 2)
23 (1 5 4 3 2)	47 (2 5 4 3 1)	71 (3 5 4 2 1)	95 (4 5 3 2 1)	119 (5 4 3 2 1)

Generating all permutations of an input list

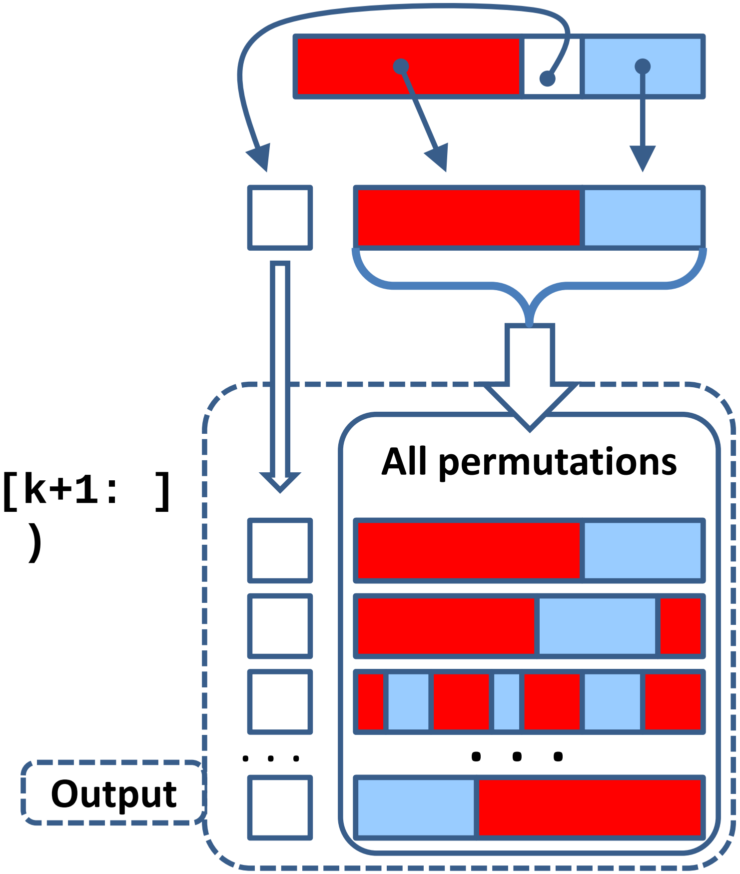
```
  # For each item in myList, run recursion on myList
  # with that item removed.
  # Prepend the removed item to each permutation returned
  # from the recursion.

def allperm ( myList ):

  if len(myList) == 1:
    return [myList]

  result = []
  for k in range( len(myList) ):
    item = myList[k]
    shorterList =  myList[ :k] + myList[k+1: ]
    shorterPerms = allperm( shorterList )
    for perm in shorterPerms:
      result.append( [item] + perm )

  return result
```



Poor time and space complexity, because of multiple lists generation.

Generating permutations in lexicographical order

Permutations of {1,2,3,4,5,6,7,8,9}

Permutation:
(5 1 8 3 9 7 6 4 2)
Next permutation:
(5 1 8 4 2 3 6 7 9)

Lexicographical order of permutations

Next permutation of
(5 1 8 3 9 7 6 4 2) :

1.
Identify last increasing neighbour pair -- 3 and 9
(5 1 8 **3 9** 7 6 4 2)

2.
Swap 3 with the smallest value bigger than 3
to the right of 3:
(5 1 8 **4** 9 7 6 **3** 2)

3. Reverse the sequence to the right of 4
(= to the right of the original position of 3)
(5 1 8 4 **2 3 6 7 9**)

Generating permutations in lexicographical order

```
def nextperm( perm ):
    n = len(perm)
    # start at the last position
    j = n-1
    # find last ascending pair
    while True:
        if j == 0:
            return False # no next permutation
        if perm[j-1] > perm[j]:
            j -= 1
        else: break
    j1 = j-1 # remember  position
    # find smallest bigger element than perm[j1] to the right of j1
    while j < n and perm[j] > perm[j1]:  j += 1
    # index of that element:
    j -= 1
    perm[j], perm[j1] = perm[j1], perm[j] # swap
    # reverse sequence to the right of j1
    j1 += 1; j = n-1
    while j1 < j:
        perm[j], perm[j1] = perm[j1], perm[j]
        j1 += 1; j -= 1
    return True
```

The rank of permutation (3,2,5,4,1) in the list of all permutations of {1,2,3,4,5} is 59.

General strategy:

1. Note that the list of all permutations is divided into a number of blocks.

2. Establish the pattern by which the permutations are divided into blocks.

3. Note that this pattern has recursive character.

4. Using the established pattern, count (recursively) the number of blocks which precede the given permutation (3,2,5,4,1) in the list of all permutations . This number is equal to the rank of the subset.

24 (2 1 3 4 5)
25 (2 1 3 5 4)
26 (2 1 4 3 5)
27 (2 1 4 5 3)
28 (2 1 5 3 4)
29 (2 1 5 4 3)

Note the regular sizing of the blocks.

Ranks of permutations

```
def rankPermutation( perm ):
    n = len( perm )
    if n == 1: return 0

    rank = (perm[0]-1) * factorial(n-1)

    # consider the permutation without the 1st element
    perm1 = perm[1:]  # copy w/o perm[0]
    # "normalize" the resulting permutation
    # for recursive processing
    for j in range(len(perm1)):
        if perm1[j] > perm[0]:
            perm1[j] -= 1

    return rank + rankPermutation( perm1 )
```

rank((5 1 8 3 9 7 6 4 2)) == 4×8! + rank ((1 7 3 8 6 5 4 2))

decreased by 1

```
def unrankPerm( rank, permLen ):
    n = permLen # just synonym
    if n == 1: return [1]  # simplest possible permutation

    # count how many blocks of size n-1
    # would fit into list of all permutations
    # before the given rank
    blocksCount = rank // factorial(n-1)  # integer div

    # construct the first element of the permutation
    firstElem = blocksCount + 1

    # calculate remaining rank to feed into recursion
    rank = rank % factorial(n-1)

    # exploit recursion
    perm = unrankPerm( rank, n-1)

    # "fit" the returned permutation to current size n
    for j in range( len(perm)):
        if perm[j] >= firstElem:
            perm[j] += 1

    return [firstElem] + perm  # concatenate lists
```

Studying subsets of {1,2,...,9}

Set: {1, 2, 3, 4, 5, 6 ,7, 8, 9}

Example subsets:

	1, 2, 3, 4, 5, 6 ,7, 8, 9	
A = {1}	$1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0_{Bin}$	$= 128_{Dec}$
A = {1, 2}	$1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0_{Bin}$	$= 192_{Dec}$
A = {2, 4, 7, 9}	$0\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 1_{Bin}$	$= 165_{Dec}$
A = { 6 }	$0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0_{Bin}$	$= 8_{Dec}$
A = { 6, 7 }	$0\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0_{Bin}$	$= 12_{Dec}$
A = { 6, 7, 8 }	$0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 0_{Bin}$	$= 14_{Dec}$
A = {1, 2, 3, 4, 5, 6 ,7, 8, 9}	$1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1_{Bin}$	$= 511_{Dec}$

Studying k-subsets of {1, 2, ..., N}

Numbers 1, 2, ..., N
can be perceived as just labels or indexes of some other items/objects x1, x2, ..., xN.

All ideas discussed here apply to the subsets of {1, 2, ..., N}
and to the subsets of {x1, x2, ..., xN}, in the same way.

Example: set {1, 2, 3, 4, 5 }

index { 1, 2, 3, 4 , 5 }
set { Ann, Bob, Don, Ema , Jan }

2-subsets of
5-element sets

{1, 2}	{Ann, Bob}
{1, 3}	{Ann, Don}
{1, 4}	{Ann, Ema}
{1, 5}	{Ann, Jan}
{2, 3}	{Bob, Don}
{2, 4}	{Bob, Ema}
{2, 5}	Bob, Jan}
{3, 4}	{Don, Ema}
{3, 5}	{Don, Jan}
{4, 5}	{Ema , Jan}

List all k-subsets of {1, 2, ..., N}

All 3-subsets of {1, 2, ..., 6}

{1, 2, 3}			
{1, 2, 4}			
{1, 2, 5}			
{1, 2, 6}			
{1, 3, 4}	{2, 3, 4}		
{1, 3, 5}	{2, 3, 5}		
{1, 3, 6}	{2, 3, 6}		
{1, 4, 5}	{2, 4, 5}	{3, 4, 5}	
{1, 4, 6}	{2, 4, 6}	{3, 4, 6}	
{1, 5, 6}	{2, 5, 6}	{3, 5, 6}	{4, 5, 6}

All 3-subsets of {1, 2, ..., 7}

{1, 2, 3}				
{1, 2, 4}				
{1, 2, 5}				
{1, 2, 6}				
{1, 2, 7}				
{1, 3, 4}	{2, 3, 4}			
{1, 3, 5}	{2, 3, 5}			
{1, 3, 6}	{2, 3, 6}			
{1, 3, 7}	{2, 3, 7}			
{1, 4, 5}	{2, 4, 5}	{3, 4, 5}		
{1, 4, 6}	{2, 4, 6}	{3, 4, 6}		
{1, 4, 7}	{2, 4, 7}	{3, 4, 7}		
{1, 5, 6}	{2, 5, 6}	{3, 5, 6}	{4, 5, 6}	
{1, 5, 7}	{2, 5, 7}	{3, 5, 7}	{4, 5, 7}	
{1, 6, 7}	{2, 6, 7}	{3, 6, 7}	{4, 6, 7}	{5, 6, 7}

Note the recursive (self-similar) structure of the listings.

List all 4-subsets of {1, 2, ..., 8 }

{1, 2, 3, 4}
{1, 2, 3, 5}
{1, 2, 3, 6}
{1, 2, 3, 7}
{1, 2, 3, 8}

{1, 2, 4, 5}
{1, 2, 4, 6}
{1, 2, 4, 7}
{1, 2, 4, 8}

{1, 2, 5, 6}
{1, 2, 5, 7}
{1, 2, 5, 8}

{1, 2, 6, 7}
{1, 2, 6, 8}

{1, 2, 7, 8}

{1, 3, 4, 5}
{1, 3, 4, 6}
{1, 3, 4, 7}
{1, 3, 4, 8}

{1, 3, 5, 6}
{1, 3, 5, 7}
{1, 3, 5, 8}

{1, 3, 6, 7}
{1, 3, 6, 8}

{1, 3, 7, 8}

{1, 4, 5, 6}
{1, 4, 5, 7}
{1, 4, 5, 8}

{1, 4, 6, 7}
{1, 4, 6, 8}

{1, 4, 7, 8}

{1, 5, 6, 7}
{1, 5, 6, 8}

{1, 5, 7, 8}

{1, 6, 7, 8}

{2, 3, 4, 5}
{2, 3, 4, 6}
{2, 3, 4, 7}
{2, 3, 4, 8}

{2, 3, 5, 6}
{2, 3, 5, 7}
{2, 3, 5, 8}

{2, 3, 6, 7}
{2, 3, 6, 8}

{2, 3, 7, 8}

{2, 4, 5, 6}
{2, 4, 5, 7}
{2, 4, 5, 8}

{2, 4, 6, 7}
{2, 4, 6, 8}

{2, 4, 7, 8}

{2, 5, 6, 7}
{2, 5, 6, 8}

{2, 5, 7, 8}

{2, 6, 7, 8}

{3, 4, 5, 6}
{3, 4, 5, 7}
{3, 4, 5, 8}

{3, 4, 6, 7}
{3, 4, 6, 8}

{3, 4, 7, 8}

{3, 5, 6, 7}
{3, 5, 6, 8}

{3, 5, 7, 8}

{3, 6, 7, 8}

{4, 5, 6, 7}
{4, 5, 6, 8}

{4, 5, 7, 8}

{4, 6, 7, 8}

{5, 6, 7, 8}

Note the recursive (self-similar) structure of the listings.

List all k-subsets of {1, 2, ..., N}

```python
# Idea:
# For each item I in the set generate all subsets
# of size k-1 using only the elements to the right of I
# (with higher index in the list)
# and prepend I to each of the generated subsets.

def k_subsets2( set, k ):

  # manage obvious edge cases
  if k == 0:
    return [[]]
  if k == len(set):
    return [set]

  # compose the result
  result = []
  for i in range( len(set) ):
    smallerSubsets = k_subsets2( set[i+1:], k-1 )
    for subset in smallerSubsets:
      result.append( [set[i]] + subset )

  return result
```

Poor time and space complexity, because of multiple lists generation.

List all k-subsets of {1, 2, ..., N}

```
# Collect the items of the subset in a single result list.
# Process lists from the end to the beginning,
# with decreasing remaining depth to simplify the code.
# No aditional index calculations! Cool, isn't it? :-)

def k_subsets3b( set, i_end, result, remainingDepth ):
  if remainingDepth < 0:     # note the zero!
    print( result )          # or add to some global variable
    return

  for i in range( i_end, remainingDepth-1, -1 ):  # go backwards
    result[remainingDepth] = set[i]
    k_subsets3b( set, i-1, result, remainingDepth-1 )


# call:
set = [ 1,2,3, ... ]
k = ...
k_subsets3b( set, len(set)-1, [0]*k, k-1 )
```

See attached allksubs.py for more code variants.

Appropriate time and space complexity.

List all k-subsets of {1, 2, ..., N} using characteristic vector

Example: All 3-subsets of {1, 2, 3, 4, 5, 6}

characteristic vector	subset	rank
1 1 1 0 0 0	[1, 2, 3]	0
1 1 0 1 0 0	[1, 2, 4]	1
1 1 0 0 1 0	[1, 2, 5]	2
1 1 0 0 0 1	[1, 2, 6]	3
1 0 1 1 0 0	[1, 3, 4]	4
1 0 1 0 1 0	[1, 3, 5]	5
1 0 1 0 0 1	[1, 3, 6]	6
1 0 0 1 1 0	[1, 4, 5]	7
1 0 0 1 0 1	[1, 4, 6]	8
1 0 0 0 1 1	[1, 5, 6]	9
0 1 1 1 0 0	[2, 3, 4]	10
0 1 1 0 1 0	[2, 3, 5]	11
0 1 1 0 0 1	[2, 3, 6]	12
0 1 0 1 1 0	[2, 4, 5]	13
0 1 0 1 0 1	[2, 4, 6]	14
0 1 0 0 1 1	[2, 5, 6]	15
0 0 1 1 1 0	[3, 4, 5]	16
0 0 1 1 0 1	[3, 4, 6]	17
0 0 1 0 1 1	[3, 5, 6]	18
0 0 0 1 1 1	[4, 5, 6]	19

Identify a k-subset by its **characteristic (indicator)** vector. That is, a 0/1 vector of length N containing exactly k 1s.

Generate all characteristic vectors, in descending lexicographical order, to represent all k-subsets, from {1, 2, ..., k} to {N-k+1, N-k+2, ..., N }, that is from (1, 1, ..., 1, 0, ..., 0, 0) to (0, 0, ..., 0, 1, ..., 1, 1).

Abstain from recursion, iteration also works in this case.

List all k-subsets of {1, 2, ..., N} using characteristic vector

```
def nextChi( chi, k, N ):
    # skip all 1s at the end:
    j1 = N-1
    while chi[j1] == 1: j1 -= 1

     # no more subsets?
    if j1 == N-k-1: return False

    # next subset
    j0 = j1-1
    while chi[j0] == 0: j0 -= 1
    chi[j0], chi[j0+1] = 0, 1  # move 1 to the right

    # move remaining 1s from the end to just behind current 1
    numOfEndOnes = N-j1-1
    for j in range( j0+2, j0+2 + numOfEndOnes ): chi[j] = 1
    for j in range( j0+2 + numOfEndOnes, N ): chi[j] = 0
    return True
```

See attached allksubs.py for more code variants.

Appropriate time and space complexity.

List all k-subsets of {1, 2, ..., N} using characteristic vector

```
def k_subsets4( set, k ):
    N = len(set)
    if k == N:
       print( *set ); return

    chi = [1]*k  + [0]*(N-k)
    rank = 0
    while True:
          print( *chi, end = '    ' )
          print( [set[k] for k in range(N) if chi[k] == 1 ], rank )
          if nextChi( chi, k, N ) == False: break
          rank += 1
```

See attached allksubs.py for more code variants.

Appropriate time and space complexity.

Studying 4-subsets of {1,2,...,12}

All 4-subsets of {1,2,...,12} are (on this and on the next 8 slides):

0 {1 2 3 4}
1 {1 2 3 5}
2 {1 2 3 6}
3 {1 2 3 7}
4 {1 2 3 8}
5 {1 2 3 9}
6 {1 2 3 10}
7 {1 2 3 11}
8 {1 2 3 12}

9 {1 2 4 5}
10 {1 2 4 6}
11 {1 2 4 7}
12 {1 2 4 8}
13 {1 2 4 9}
14 {1 2 4 10}
15 {1 2 4 11}
16 {1 2 4 12}

17 {1 2 5 6}
18 {1 2 5 7}
19 {1 2 5 8}
20 {1 2 5 9}
21 {1 2 5 10}
22 {1 2 5 11}
23 {1 2 5 12}

24 {1 2 6 7}
25 {1 2 6 8}
26 {1 2 6 9}
27 {1 2 6 10}
28 {1 2 6 11}
29 {1 2 6 12}

30 {1 2 7 8}
31 {1 2 7 9}
32 {1 2 7 10}
33 {1 2 7 11}
34 {1 2 7 12}

35 {1 2 8 9}
36 {1 2 8 10}
37 {1 2 8 11}
38 {1 2 8 12}

39 {1 2 9 10}
40 {1 2 9 11}
41 {1 2 9 12}

42 {1 2 10 11}
43 {1 2 10 12}

44 {1 2 11 12}

45 {1 3 4 5}
46 {1 3 4 6}
47 {1 3 4 7}
48 {1 3 4 8}
49 {1 3 4 9}
50 {1 3 4 10}
51 {1 3 4 11}
52 {1 3 4 12}

53 {1 3 5 6}
54 {1 3 5 7}
55 {1 3 5 8}
56 {1 3 5 9}
57 {1 3 5 10}
58 {1 3 5 11}
59 {1 3 5 12}

60 {1 3 6 7}
61 {1 3 6 8}
62 {1 3 6 9}
63 {1 3 6 10}
64 {1 3 6 11}
65 {1 3 6 12}

66 {1 3 7 8}
67 {1 3 7 9}
68 {1 3 7 10}
69 {1 3 7 11}
70 {1 3 7 12}

71 {1 3 8 9}
72 {1 3 8 10}
73 {1 3 8 11}
74 {1 3 8 12}

Studying 4-subsets of {1,2,...,12}

The vertical spaces remind about the regularity patterns in the list.

75 {1 3 9 10}
76 {1 3 9 11}
77 {1 3 9 12}

78 {1 3 10 11}
79 {1 3 10 12}

80 {1 3 11 12}

81 {1 4 5 6}
82 {1 4 5 7}
83 {1 4 5 8}
84 {1 4 5 9}
85 {1 4 5 10}
86 {1 4 5 11}
87 {1 4 5 12}

88 {1 4 6 7}
89 {1 4 6 8}
90 {1 4 6 9}
91 {1 4 6 10}
92 {1 4 6 11}
93 {1 4 6 12}

94 {1 4 7 8}
95 {1 4 7 9}
96 {1 4 7 10}
97 {1 4 7 11}
98 {1 4 7 12}

99 {1 4 8 9}
100 {1 4 8 10}
101 {1 4 8 11}
102 {1 4 8 12}

103 {1 4 9 10}
104 {1 4 9 11}
105 {1 4 9 12}
106 {1 4 10 11}
107 {1 4 10 12}
108 {1 4 11 12}

109 {1 5 6 7}
110 {1 5 6 8}
111 {1 5 6 9}
112 {1 5 6 10}
113 {1 5 6 11}
114 {1 5 6 12}

115 {1 5 7 8}
116 {1 5 7 9}
117 {1 5 7 10}
118 {1 5 7 11}
119 {1 5 7 12}

120 {1 5 8 9}
121 {1 5 8 10}
122 {1 5 8 11}
123 {1 5 8 12}

124 {1 5 9 10}
125 {1 5 9 11}
126 {1 5 9 12}

127 {1 5 10 11}
128 {1 5 10 12}

129 {1 5 11 12}

Studying 4-subsets of {1,2,...,12}

130 {1 6 7 8}
131 {1 6 7 9}
132 {1 6 7 10}
133 {1 6 7 11}
134 {1 6 7 12}

135 {1 6 8 9}
136 {1 6 8 10}
137 {1 6 8 11}
138 {1 6 8 12}

139 {1 6 9 10}
140 {1 6 9 11}
141 {1 6 9 12}

142 {1 6 10 11}
143 {1 6 10 12}

144 {1 6 11 12}

145 {1 7 8 9}
146 {1 7 8 10}
147 {1 7 8 11}
148 {1 7 8 12}

149 {1 7 9 10}
150 {1 7 9 11}
151 {1 7 9 12}

152 {1 7 10 11}
153 {1 7 10 12}

154 {1 7 11 12}

155 {1 8 9 10}
156 {1 8 9 11}
157 {1 8 9 12}

158 {1 8 10 11}
159 {1 8 10 12}

160 {1 8 11 12}

161 {1 9 10 11}
162 {1 9 10 12}

163 {1 9 11 12}

164 {1 10 11 12}

165 {2 3 4 5}
166 {2 3 4 6}
167 {2 3 4 7}
168 {2 3 4 8}
169 {2 3 4 9}
170 {2 3 4 10}
171 {2 3 4 11}
172 {2 3 4 12}

173 {2 3 5 6}
174 {2 3 5 7}
175 {2 3 5 8}
176 {2 3 5 9}
177 {2 3 5 10}
178 {2 3 5 11}
179 {2 3 5 12}

Studying 4-subsets of {1,2,...,12}

180 {2 3 6 7}
181 {2 3 6 8}
182 {2 3 6 9}
183 {2 3 6 10}
184 {2 3 6 11}
185 {2 3 6 12}

186 {2 3 7 8}
187 {2 3 7 9}
188 {2 3 7 10}
189 {2 3 7 11}
190 {2 3 7 12}

191 {2 3 8 9}
192 {2 3 8 10}
193 {2 3 8 11}
194 {2 3 8 12}

195 {2 3 9 10}
196 {2 3 9 11}
197 {2 3 9 12}

198 {2 3 10 11}
199 {2 3 10 12}
200 {2 3 11 12}

201 {2 4 5 6}
202 {2 4 5 7}
203 {2 4 5 8}
204 {2 4 5 9}
205 {2 4 5 10}
206 {2 4 5 11}
207 {2 4 5 12}

208 {2 4 6 7}
209 {2 4 6 8}
210 {2 4 6 9}
211 {2 4 6 10}
212 {2 4 6 11}
213 {2 4 6 12}

214 {2 4 7 8}
215 {2 4 7 9}
216 {2 4 7 10}
217 {2 4 7 11}
218 {2 4 7 12}

219 {2 4 8 9}
220 {2 4 8 10}
221 {2 4 8 11}
222 {2 4 8 12}

223 {2 4 9 10}
224 {2 4 9 11}
225 {2 4 9 12}

226 {2 4 10 11}
227 {2 4 10 12}
228 {2 4 11 12}

229 {2 5 6 7}
230 {2 5 6 8}
231 {2 5 6 9}
232 {2 5 6 10}
233 {2 5 6 11}
234 {2 5 6 12}

Studying 4-subsets of {1,2,...,12}

235 {2 5 7 8}
236 {2 5 7 9}
237 {2 5 7 10}
238 {2 5 7 11}
239 {2 5 7 12}

240 {2 5 8 9}
241 {2 5 8 10}
242 {2 5 8 11}
243 {2 5 8 12}

244 {2 5 9 10}
245 {2 5 9 11}
246 {2 5 9 12}

247 {2 5 10 11}
248 {2 5 10 12}

249 {2 5 11 12}

250 {2 6 7 8}
251 {2 6 7 9}
252 {2 6 7 10}
253 {2 6 7 11}
254 {2 6 7 12}

255 {2 6 8 9}
256 {2 6 8 10}
257 {2 6 8 11}
258 {2 6 8 12}

259 {2 6 9 10}
260 {2 6 9 11}
261 {2 6 9 12}

262 {2 6 10 11}
263 {2 6 10 12}

264 {2 6 11 12}

265 {2 7 8 9}
266 {2 7 8 10}
267 {2 7 8 11}
268 {2 7 8 12}

269 {2 7 9 10}
270 {2 7 9 11}
271 {2 7 9 12}

272 {2 7 10 11}
273 {2 7 10 12}

274 {2 7 11 12}

275 {2 8 9 10}
276 {2 8 9 11}
277 {2 8 9 12}

278 {2 8 10 11}
279 {2 8 10 12}

280 {2 8 11 12}

281 {2 9 10 11}
282 {2 9 10 12}

283 {2 9 11 12}

284 {2 10 11 12}

285 {3 4 5 6}
286 {3 4 5 7}
287 {3 4 5 8}
288 {3 4 5 9}
289 {3 4 5 10}
290 {3 4 5 11}
291 {3 4 5 12}

Studying 4-subsets of {1,2,...,12}

292 {3 4 6 7}
293 {3 4 6 8}
294 {3 4 6 9}
295 {3 4 6 10}
296 {3 4 6 11}
297 {3 4 6 12}

298 {3 4 7 8}
299 {3 4 7 9}
300 {3 4 7 10}
301 {3 4 7 11}
302 {3 4 7 12}

303 {3 4 8 9}
304 {3 4 8 10}
305 {3 4 8 11}
306 {3 4 8 12}

307 {3 4 9 10}
308 {3 4 9 11}
309 {3 4 9 12}

310 {3 4 10 11}
311 {3 4 10 12}

312 {3 4 11 12}

313 {3 5 6 7}
314 {3 5 6 8}
315 {3 5 6 9}
316 {3 5 6 10}
317 {3 5 6 11}
318 {3 5 6 12}

319 {3 5 7 8}
320 {3 5 7 9}
321 {3 5 7 10}
322 {3 5 7 11}
323 {3 5 7 12}

324 {3 5 8 9}
325 {3 5 8 10}
326 {3 5 8 11}
327 {3 5 8 12}

328 {3 5 9 10}
329 {3 5 9 11}
330 {3 5 9 12}

331 {3 5 10 11}
332 {3 5 10 12}

333 {3 5 11 12}

334 {3 6 7 8}
335 {3 6 7 9}
336 {3 6 7 10}
337 {3 6 7 11}
338 {3 6 7 12}

339 {3 6 8 9}
340 {3 6 8 10}
341 {3 6 8 11}
342 {3 6 8 12}

343 {3 6 9 10}
344 {3 6 9 11}
345 {3 6 9 12}

346 {3 6 10 11}
347 {3 6 10 12}

348 {3 6 11 12}

Studying 4-subsets of {1,2,...,12}

349 {3 7 8 9}
350 {3 7 8 10}
351 {3 7 8 11}
352 {3 7 8 12}

353 {3 7 9 10}
354 {3 7 9 11}
355 {3 7 9 12}

356 {3 7 10 11}
357 {3 7 10 12}

358 {3 7 11 12}

359 {3 8 9 10}
360 {3 8 9 11}

361 {3 8 9 12}
362 {3 8 10 11}
363 {3 8 10 12}

364 {3 8 11 12}

365 {3 9 10 11}
366 {3 9 10 12}

367 {3 9 11 12}

368 {3 10 11 12}

369 {4 5 6 7}
370 {4 5 6 8}
371 {4 5 6 9}
372 {4 5 6 10}
373 {4 5 6 11}
374 {4 5 6 12}

375 {4 5 7 8}
376 {4 5 7 9}
377 {4 5 7 10}
378 {4 5 7 11}
379 {4 5 7 12}

380 {4 5 8 9}
381 {4 5 8 10}
382 {4 5 8 11}
383 {4 5 8 12}

384 {4 5 9 10}
385 {4 5 9 11}
386 {4 5 9 12}

387 {4 5 10 11}
388 {4 5 10 12}

389 {4 5 11 12}

390 {4 6 7 8}
391 {4 6 7 9}
392 {4 6 7 10}
393 {4 6 7 11}
394 {4 6 7 12}

395 {4 6 8 9}
396 {4 6 8 10}
397 {4 6 8 11}
398 {4 6 8 12}

399 {4 6 9 10}
400 {4 6 9 11}
401 {4 6 9 12}

402 {4 6 10 11}
403 {4 6 10 12}

404 {4 6 11 12}

Studying 4-subsets of {1,2,...,12}

405 {4 7 8 9}
406 {4 7 8 10}
407 {4 7 8 11}
408 {4 7 8 12}

409 {4 7 9 10}
410 {4 7 9 11}
411 {4 7 9 12}

412 {4 7 10 11}
413 {4 7 10 12}

414 {4 7 11 12}

415 {4 8 9 10}
416 {4 8 9 11}
417 {4 8 9 12}

418 {4 8 10 11}
419 {4 8 10 12}

420 {4 8 11 12}

421 {4 9 10 11}
422 {4 9 10 12}

423 {4 9 11 12}

424 {4 10 11 12}

425 {5 6 7 8}
426 {5 6 7 9}
427 {5 6 7 10}
428 {5 6 7 11}
429 {5 6 7 12}

430 {5 6 8 9}
431 {5 6 8 10}
432 {5 6 8 11}
433 {5 6 8 12}

434 {5 6 9 10}
435 {5 6 9 11}
436 {5 6 9 12}

437 {5 6 10 11}
438 {5 6 10 12}

439 {5 6 11 12}

440 {5 7 8 9}
441 {5 7 8 10}
442 {5 7 8 11}
443 {5 7 8 12}

444 {5 7 9 10}
445 {5 7 9 11}
446 {5 7 9 12}

447 {5 7 10 11}
448 {5 7 10 12}

449 {5 7 11 12}

450 {5 8 9 10}
451 {5 8 9 11}
452 {5 8 9 12}

453 {5 8 10 11}
454 {5 8 10 12}

455 {5 8 11 12}

Studying 4-subsets of {1,2,...,12}

456 {5 9 10 11}
457 {5 9 10 12}

458 {5 9 11 12}

459 {5 10 11 12}

460 {6 7 8 9}
461 {6 7 8 10}
462 {6 7 8 11}
463 {6 7 8 12}

464 {6 7 9 10}
465 {6 7 9 11}
466 {6 7 9 12}

467 {6 7 10 11}
468 {6 7 10 12}
469 {6 7 11 12}

470 {6 8 9 10}
471 {6 8 9 11}
472 {6 8 9 12}

473 {6 8 10 11}
474 {6 8 10 12}

475 {6 8 11 12}

476 {6 9 10 11}
477 {6 9 10 12}

478 {6 9 11 12}

479 {6 10 11 12}

480 {7 8 9 10}
481 {7 8 9 11}
482 {7 8 9 12}

483 {7 8 10 11}
484 {7 8 10 12}

485 {7 8 11 12}

486 {7 9 10 11}
487 {7 9 10 12}

488 {7 9 11 12}

489 {7 10 11 12}

490 {8 9 10 11}
491 {8 9 10 12}

492 {8 9 11 12}

493 {8 10 11 12}

494 {9 10 11 12}

The rank of subset {6,8,9,11} in all 4-subsets of {1,2,...,12} is 471 (see previous slide).

General strategy:

1. Note that the list of all 4-subsets is divided into a number of blocks.

2. Establish the pattern by which the 4-subsets are divided into blocks.

3. Note that this pattern has recursive character.

4. Using the established pattern, count (recursively) the number of blocks
(on all significant recursion levels)
which precede the given subset {6,8,9,11} in the list of all subsets.
This number is equal to the rank of the subset.

The rank of subset {6,8,9,11} in all 4-subsets of {1,2,...,12} is 471 (see earlier slide).

The minimum item in {6,8,9,11} is 6.
Therefore {6,8,9,11} is preceded in the list by all 4- subsets which contain values
-- 1 and bigger
-- 2 and bigger
-- 3 and bigger
-- 4 and bigger
-- 5 and bigger

Specifically, those are:

0 {1 2 3 4}	**165 {2 3 4 5}**	**285 {3 4 5 6}**	**369 {4 5 6 7}**	**425 {5 6 7 8}**
1 {1 2 3 5}	**166 {2 3 4 6}**	**286 {3 4 5 7}**	**370 {4 5 6 8}**	**426 {5 6 7 9}**
...	**...**	**...**	**...**	**...**
...	**...**	**...**	**...**	**...**
164 {1 10 11 12}	**284 {2 10 11 12}**	**368 {3 10 11 12}**	**424 {4 10 11 12}**	**459 {5 10 11 12}**

The size of each of these 5 blocks is computed on next two slides.

Spoiler: *Each size is a binomial coefficient.*

Studying 4-subsets of {1,2,...,12}

Level 0

The rank of subset {6,8,9,11} in all 4-subsets of {1,2,...,12} is 471 (see earlier slide).

0 {1 2 3 4}
1 {1 2 3 5}
...
...
164 {1 10 11 12}

Block 1. All 4-subsets which contain 1 and bigger values.
Value 1 is present in all subsets in the block.
When we remove 1 from each item in the block, we find that the size of the block is equal to the number of all 3-subsets of the set {2,3,4,...,12}.
Formally, that number is the same as the number of all 3-subsets of the set {1,2,3,...,11} *).

And that, in turn, is equal to binCoeff(11,3) = 11!/(3!*8!) = 165

165 {2 3 4 5}
166 {2 3 4 6}
...
...
284 {2 10 11 12}

Block 2. All 4-subsets which contain 2 and bigger values.
Value 2 is present in all subsets in the block.
When we remove 2 from each item in the block, we find that the size of the block is equal to the number of all 3-subsets of the set {3,4,5,...,12}.
Formally, that number is the same as the number of all 3-subsets of the set {1,2,3,...,10}.

And that, in turn, is equal to binCoeff(10,3) = 10!/(3!*7!) = 120

*) Should be obvious, as the size of sets {2,3,4,...,12} and {1,2,3,...,11} is clearly the same.

Studying 4-subsets of {1,2,...,12}

Level 0

The rank of subset {6,8,9,11} in all 4-subsets of {1,2,...,12} is 471 (see earlier slide).

285 {3 4 5 6}
286 {3 4 5 7}
...
...
368 {3 10 11 12}

Block 3. All 4-subsets which contain 3 and bigger values.
The size of the block is equal to the number
of all 3-subsets of the set {4,5,6,...,12}.
Formally, that number is the same as the number
of all 3-subsets of the set {1,2,3,...,9}.

And that, in turn, is equal to binCoeff(9,3) = 9!/(3!*6!) = 84

369 {4 5 6 7}
370 {4 5 6 8}
...
...
424 {4 10 11 12}

Block 4. All 4-subsets which contain 4 and bigger values.
Formally, the number of those subsets is the same as the number
of all 3-subsets of the set {1,2,3,...,8}.

And that is equal to binCoeff(8,3) = 8!/(3!*5!) = 56

425 {5 6 7 8}
426 {5 6 7 9}
...
...
459 {5 10 11 12}

Block 5. All 4-subsets which contain 5 and bigger values.
Formally, the number of those subsets is the same as the number
of all 3-subsets of the set {1,2,3,...,7}.

And that is equal to binCoeff(7,3) = 7!/(3!*4!) = 35

Studying 4-subsets of {1,2,...,12}

Level 0

The rank of subset {6,8,9,11} in all 4-subsets of {1,2,...,12} is 471.

The subset {6,8,9,11} is preceded by 5 blocks which total size is

165 + 120 + 84 + 56 + 35 = 460.

Thus, the rank of {6,8,9,11} is 460 or bigger.

The 4-subset {6,8,9,11} is itself in the block 6 which contains values 6 and higher:

460 {6 7 8 9}
461 {6 7 8 10}
...
...
479 {6 10 11 12}

The rank of {6,8,9,11} in all 4-subsets of {1,2,...,12} is equal to
460 + the rank of {6,8,9,11} in all 4-subsets of {6,7,8,...,12}.

Note that the value 6 is common in all subsets in this block.
Remove it from the subsets and from the set {6,7,8,...,12}.

Now, recursion kicks in.

Therefore:

A. The rank of {6,8,9,11} in all 4-subsets of {6,7,8,...,12} is equal to
the rank of {8,9,11} in all 3-subsets of {7,8,...,12}.

B. The rank of {8,9,11} in all 3-subsets of {7,8,...,12} is equal to
the rank of {2,3,5} in all 3-subsets of {1,2,...,6}.
(just formally subtract 6 from all elements in the subset and the set {7,8,...,12})

Studying 4-subsets of {1,2,...,12}

Level 0

The rank of subset {6,8,9,11} in all 4-subsets of {1,2,...,12} is 471.

The subset {6,8,9,11} is preceded by 5 blocks which total size is
165 + 120 + 84 + 56 + 35 = 460.
Thus, the rank of {6,8,9,11} is 460 or bigger.

The 4-subset {6,8,9,11} is itself in the block 6 which contains values 6 and higher:

460 {6 7 8 9}
461 {6 7 8 10}
...
...
479 {6 10 11 12}

Previous slide A+B:
The rank of {6,8,9,11} in all 4-subsets of {6,7,8,...,12} is equal to the rank of {2,3,5} in all 3-subsets of {1,2,...,6}.

Apply recursion -- same problem structure, smaller parameters.

The rank of {2,3,5} in all 3-subsets of {1,2,...,6} is 11.

Studying 3-subsets of {1,2,...,6}

Level 1

All 3-subsets of {1,2,...,6} are

0 {1 2 3}	10 {2 3 4}	16 {3 4 5}	19 {4 5 6}
1 {1 2 4}	**11 {2 3 5}**	17 {3 4 6}	
2 {1 2 5}	12 {2 3 6}		
3 {1 2 6}		18 {3 5 6}	
	13 {2 4 5}		
4 {1 3 4}	14 {2 4 6}		
5 {1 3 5}			
6 {1 3 6}	15 {2 5 6}		
7 {1 4 5}			
8 {1 4 6}			
9 {1 5 6}			

Studying 3-subsets of {1,2,...,12}

Level 1

The rank of subset {2,3,5} in all 3-subsets of {1,2,...,6} is 11 (see previous slide).

The minimum item in {2,3,5} is 1.
Therefore {2,3,5} is preceded in the list by all 3-subsets which contain values
-- 1 and bigger

Specifically, that is:

0 {1 2 3}
1 {1 2 4}
...
9 {1 5 6}

Block 1. All 3-subsets which contain 1 and bigger values.
Value 1 is present in all subsets in the block.
When we remove 1 from each item in the block, we find that the size of the block is equal to the number of all 2-subsets of the set {2,3,4,...,6}.
Formally, that number is the same as the number of all 2-subsets of the set {1,2,3,...,5}.
And that, in turn, is equal to binCoeff(5,2) = 5!/(2!*3!) = 10

Level 1

The rank of subset {2,3,5} in all 3-subsets of {1,2,...,6} is 11.

The subset {6,8,9,11} is preceded by 1 blocks which size is10.
Thus, the rank of {2,3,5} is 10 or bigger.

The 3-subset {2,3,5} is itself in the block 2 which contains values 2 and higher:

460 {6 7 8 9}
461 {6 7 8 10}
...
...
479 {6 10 11 12}

The rank of {2,3,5} in all 3-subsets of {1,2,...,6} is equal to
10 + the rank of {2,3,5} in all 4-subsets of {2,3,4,...,6}.

Note that the value 2 is common in all subsets in this block.
Remove it from the subsets and from the set {2,3,4,...,6}.

Therefore:

A. The rank of {2,3,5} in all 3-subsets of {2,3,4,...,6} is equal to
the rank of {3,5} in all 2-subsets of {3,4,...,6}.

B. The rank of {3,5} in all 2-subsets of {3,4,...,6} is equal to
the rank of {1 3} in all 2-subsets of {1,2,...,4}.
(Just formally subtract 2 from all elements in the subset and the set {3,4,...,6}.)

Level 2

All 2-subsets of {1,2,...,4} are

0 {1 2}
1 {1 3}
2 {1 4}

3 {2 3}
4 {2 4}

5 {3 4}

The rank of subset {1,3} in all 2-subsets of {1,2,...,4} is 1.

0 {1 2}
1 {1 3}
...
2 {1 4}

The minimum item in {1, 3} is 1.
Therefore {1,3} is in the list, in the first block.
In other words, it is preceded by 0 blocks which contain value 0 and higher.
(Value 0 cannot appear in the subset).

The rank of {1,3} in the first block in the list of all 2-subsets of {1,2,...,4} is equal to the rank of {3} in the list of all 1-subsets of {2,...,4}.
That is the same as the rank of {2} in the list of all 1-subsets of {1,...,3}.
(Just subtract 1 from all elements in the subset and the set {1,...,3}.)

Studying 1-subsets of {1,2,...,4}

Level 3

All 1-subsets of {1,2,...,3} are

0 {1}
1 {2}
2 {3}

The rank of subset {2} in all 1-subsets of {1,2,...,3} is 1.

Finding the rank of 1-element subset {a} of the set {1,2,...,X) is easy, just return a–1.

To conclude:

Finding the rank of a subset required computing recursively the size of blocks which preceded the given subset in the lexicographically ordered list of all subsets of the given size.

The computations on consecutive levels of recursion yielded total sizes
460 + 10 + 0 + 1 = 471.

```
def rankSubset( subset, n ):
    k = len(subset)
    if k == 1: return subset[0] - 1

    rank = 0
    # total number of all subsets containing
    # values 1, 2, ..., subset[0]-1, which precede the given subset
    # in the list of all subsets lexicographically sorted
    for i in range(1, subset[0]):
        rank += binCoeff( n-i, k-1 )

    # exclude first elem from the subset array
    # and "normalize" the input for recursion
    subset1 = subset[1:] # copy of subset[1..k]
    for j in range( len(subset1) ):
        subset1[j] -= subset[0]
    n1 = n - (subset[0])

    # and recurse
    return rank + rankSubset( subset1, n1)
```

```python
def unrankSubset ( rank, n, k ):
    if k == 1:  return [rank+1]   # list with single value

    # jump over appropriate number of blocks
    # which precede the subset with the given rank
    # and simultaneously construct value subset[0]
    n1 = n-1        # next n value in recursive call
    subset0 = 1
    while True:
        blockSize = binCoeff( n1, k-1 )
        if blockSize <= rank:
            rank -= blockSize
            subset0 += 1
            n1 -= 1
        else: break

    subsetRec = unrankSubset ( rank, n1, k-1 )
    for j in range( len(subsetRec) ):
        subsetRec[j] += subset0

    return [subset0] + subsetRec  # list concatenation
```

Gray code examples

n=1	n=2	n=3	n=4	n=5
0	0 0	0 0 0	0 0 0 0	0 0 0 0 0
1	0 1	0 0 1	0 0 0 1	0 0 0 0 1
	1 1	0 1 1	0 0 1 1	0 0 0 1 1
	1 0	0 1 0	0 0 1 0	0 0 0 1 0
		1 1 0	0 1 1 0	0 0 1 1 0
		1 1 1	0 1 1 1	0 0 1 1 1
		1 0 1	0 1 0 1	0 0 1 0 1
		1 0 0	0 1 0 0	0 0 1 0 0
			1 1 0 0	0 1 1 0 0
			1 1 0 1	0 1 1 0 1
			1 1 1 1	0 1 1 1 1
			1 1 1 0	0 1 1 1 0
			1 0 1 0	0 1 0 1 0
			1 0 1 1	0 1 0 1 1
			1 0 0 1	0 1 0 0 1
			1 0 0 0	0 1 0 0 0
				1 1 0 0 0
				1 1 0 0 1
				1 1 0 1 1
				1 1 0 1 0
				1 1 1 1 0
				1 1 1 1 1
				1 1 1 0 1
				1 1 1 0 0
				1 0 1 0 0
				1 0 1 0 1
				1 0 1 1 1
				1 0 1 1 0
				1 0 0 1 0
				1 0 0 1 1
				1 0 0 0 1
				1 0 0 0 0

```python
def grayCode( n ):
    if n == 1: return ["0 ", "1 "]
    gc0 = grayCode(n-1)
    gc1 = list(reversed(gc0))
    for i in range(len(gc0)):
        gc0[i] = "0 "+gc0[i]
    for i in range(len(gc1)):
        gc1[i] = "1 "+gc1[i]

    return gc0+gc1


for i in range (1,6):
    for z in grayCode( i ):
        print(z)
    print()
```