Computation of OEIS sequences A334254 and A334255 for n=6 and checking their values for n=3,4,5.

1. Main code.

In [1]:
```
%load_ext Cython
```

In [2]:
```
%%cython --annotate
from math import log

cpdef ProcessRT(t,int i,int l,int nn1):
    """This function recursively extends an input tuple of integers
    a set of sets (more precisily, a formal context), by adding to
    The length of a terminal tuple is constrained by l, while nn1 s
    The function checks whether the new tuple forms a unique closur
    processed tuples as keys and values 1 for clousre system with T

    count={}
    t=t[:]
    cdef int P
    cdef int lent=len(t)


    #checking whether the context is reducible
    for j in range(lent-1,-1,-1):
        P=i
        #print("t[j],i",t[j],i,t[j]&i,)
        tji=t[j]&i
        if tji in t[0:j]:
            return count

        if tji==t[j]:
            #print("t[j],i",t[j],i,t[j]&i,)
            for h in range(j+1,lent):
                if t[j]&t[h]==t[j]:
                    P=P&t[h]
                    if P==t[j]:
                        return count
    t.append(i)

    #checking T1 property
    if Check(t,nn1):
        count[(lent+1,t[0])]=1
    else:
        count[(lent+1,t[0])]=0

    #recursive call
    cdef unsigned long long r
    #print(l)
```

```
    if l!=0:
        r=0
        for k in range(i+1,nn1):
            #print("Process(t,k,l-1,tuples,nn1)")
            #print("t",t,"k",k)
            ucount=ProcessRT(t,k,l-1,nn1)
            for key in ucount:
                if key in count:
                    count[key]+=ucount[key]
                else:
                    count[key]=ucount[key]
            #if r<0: print("r",r)
        return count
    else:
        #print(tuples)
        #tuples.append(tuple(t))


        print("Branch process",t[0])
        return count


def isClosed(t,g):
    """Checks whether g is closed in t."""
    cont=[]
    for h in t:
        if g&h==g:
            cont.append(h)
    if len(cont)>0:
        res=cont[0]
        for h in cont:
            res=h&res
        return res==g
    else:
        return g==0


def Check(t,nn1):
    """Checks whether T1 is fulfilled for t."""
    n=int(log(nn1+1,2))
    members=[0]
    members.extend([2**i for i in range(n)])
    for m in members:
        if not isClosed(t,m):
            return False
    return True
```

Out[2]:

Generated by Cython 0.29.24

Yellow lines hint at Python interaction.
Click on a line that starts with a " + " to see the C code that Cython generated for it.

```
+01: from math import log
 02:
+03: cpdef ProcessRT(t,int i,int l,int nn1):
```

```
 04:     """ This function recursively extends an input tuple of i
ntegers, which represents
 05:     a set of sets (more precisily, a formal context), by addi
ng to it an integer i (a new set).
 06:     The length of a terminal tuple is constrained by l, while
nn1 should be passed as 2**n-1.
 07:     The function checks whether the new tuple forms a unique
closure system and returns the dictionary of the first elements of
 08:     processed tuples as keys and values 1 for clousre system
with T1 propery and 0 otherwise."""
 09:
+10:     count={}
+11:     t=t[:]
 12:     cdef int P
+13:     cdef int lent=len(t)
 14:
 15:
 16:     #checking whether the context is reducible
+17:     for j in range(lent-1,-1,-1):
+18:         P=i
 19:         #print("t[j],i",t[j],i,t[j]&i,)
+20:         tji=t[j]&i
+21:         if tji in t[0:j]:
+22:             return count
 23:
+24:         if tji==t[j]:
 25:             #print("t[j],i",t[j],i,t[j]&i,)
+26:             for h in range(j+1,lent):
+27:                 if t[j]&t[h]==t[j]:
+28:                     P=P&t[h]
+29:                     if P==t[j]:
+30:                         return count
+31:     t.append(i)
 32:
 33:     #checking T1 property
+34:     if Check(t,nn1):
+35:         count[(lent+1,t[0])]=1
 36:     else:
+37:         count[(lent+1,t[0])]=0
 38:
 39:     #recursive call
 40:     cdef unsigned long long r
 41:     #print(l)
+42:     if l!=0:
+43:         r=0
+44:         for k in range(i+1,nn1):
 45:             #print("Process(t,k,l-1,tuples,nn1)")
 46:             #print("t",t,"k",k)
+47:             ucount=ProcessRT(t,k,l-1,nn1)
+48:             for key in ucount:
+49:                 if key in count:
+50:                     count[key]+=ucount[key]
```

```
 51:                 else:
+52:                     count[key]=ucount[key]
 53:             #if r<0: print("r",r)
+54:         return count
 55:     else:
 56:         #print(tuples)
 57:         #tuples.append(tuple(t))
 58:
 59:
+60:         print("Branch process",t[0])
+61:         return count
 62:
 63:
+64: def isClosed(t,g):
 65:     """Checks whether g is closed in t."""
+66:     cont=[]
+67:     for h in t:
+68:         if g&h==g:
+69:             cont.append(h)
+70:     if len(cont)>0:
+71:         res=cont[0]
+72:         for h in cont:
+73:             res=h&res
+74:         return res==g
 75:     else:
+76:         return g==0
 77:
 78:
+79: def Check(t,nn1):
 80:     """Checks whether T1 is fulfilled for t."""
+81:     n=int(log(nn1+1,2))
+82:     members=[0]
+83:     members.extend([2**i for i in range(n)])
+84:     for m in members:
+85:         if not isClosed(t,m):
+86:             return False
+87:     return True
```

In [3]:

```
def merge(r):
    """This is a value collector for the resulting list of dictiona
    """
    merged={}
    for dic in r:
        for key in dic:
            if key in merged:
                merged[key]+=dic[key]
            else:
                merged[key]=dic[key]
    return merged
```

2. Counting A334254 for n=6.

In [28]:
```
%%time



from multiprocess import Pool
from multiprocess import cpu_count


n=6

#computing A334254 for n=6 by levels
if __name__ == "__main__":
    pool = Pool(cpu_count())

    nn1=2**n-1

    lt=[]
    lk=[]

    count={}


    for t in range(nn1):

        for i in range(t+1,nn1):
            lt.append([t])
            lk.append(i)

    ln_2=[nn1-1]* len(lt)
    lnn1=[nn1]* len(lt)



    print(list(zip(lt,lk,ln_2,lnn1)))


    #parallel execution of ProcessRT function
    res6 = pool.starmap_async(ProcessRT, zip(lt,lk,ln_2,lnn1))
    print(res6.get())# print the list of resulting dictionaries
    print(sum(merge(res6.get()).values()))#print the number of clos
    pool.close()  # 'TERM'
    pool.join()   # 'KILL'
```

```
0, (11, 13): 0, (12, 13): 0, (13, 13): 0, (14, 13): 0}, {(2, 13)
: 0, (3, 13): 0, (4, 13): 0, (5, 13): 0, (6, 13): 0, (7, 13): 0,
(8, 13): 0, (9, 13): 0, (10, 13): 0, (11, 13): 0, (12, 13): 0, (
13, 13): 0, (14, 13): 0}, {(2, 13): 0, (3, 13): 0, (4, 13): 0, (
5, 13): 0, (6, 13): 0, (7, 13): 0, (8, 13): 0, (9, 13): 0, (10,
13): 0, (11, 13): 0, (12, 13): 0, (13, 13): 0}, {(2, 13): 0, (3,
13): 0, (4, 13): 0, (5, 13): 0, (6, 13): 0, (7, 13): 0, (8, 13):
0, (9, 13): 0, (10, 13): 0, (11, 13): 0}, {(2, 13): 0, (3, 13):
0, (4, 13): 0, (5, 13): 0, (6, 13): 0, (7, 13): 0, (8, 13): 0, (
9, 13): 0, (10, 13): 0, (11, 13): 0, (12, 13): 0, (13, 13): 0},
```

```
{(2, 13): 0, (3, 13): 0, (4, 13): 0, (5, 13): 0, (6, 13): 0, (7,
13): 0, (8, 13): 0, (9, 13): 0, (10, 13): 0, (11, 13): 0, (12, 1
3): 0, (13, 13): 0}, {(2, 13): 0, (3, 13): 0, (4, 13): 0, (5, 13
): 0, (6, 13): 0, (7, 13): 0, (8, 13): 0, (9, 13): 0, (10, 13):
0, (11, 13): 0, (12, 13): 0}, {(2, 13): 0, (3, 13): 0, (4, 13):
0, (5, 13): 0, (6, 13): 0, (7, 13): 0, (8, 13): 0, (9, 13): 0, (
10, 13): 0, (11, 13): 0}, {(2, 13): 0, (3, 13): 0, (4, 13): 0, (
5, 13): 0, (6, 13): 0, (7, 13): 0, (8, 13): 0, (9, 13): 0, (10,
13): 0, (11, 13): 0}, {(2, 13): 0, (3, 13): 0, (4, 13): 0, (5, 1
```

In [31]:
```
#saving the result to A334254_6.pickle file with pickle
import pickle
pikd = open("A334254_6.pickle", "wb")
pickle.dump(res6.get(), pikd)
pikd.close()
```

In [36]:
```
res6.get()
```

```
  (12, 18): 0,
  (13, 18): 0},
 {(2, 18): 0,
  (3, 18): 0,
  (4, 18): 0,
  (5, 18): 0,
  (6, 18): 0,
  (7, 18): 0,
  (8, 18): 0,
  (9, 18): 0,
  (10, 18): 0,
  (11, 18): 0,
  (12, 18): 0,
  (13, 18): 0},
 {(2, 18): 0,
  (3, 18): 0,
  (4, 18): 0,
  (5, 18): 0,
  (6, 18): 0,
```

In [38]:
```
sum(merge(res6.get()).values())# the computed value A334254 for n=6
```

Out[38]:
```
66960965307
```

3. Checking A334254 for n=3, 4, 5.

In [5]:
```
```

```
%%time



from multiprocess import Pool
from multiprocess import cpu_count


n=5

#computing A334254 for n=5 by levels
if __name__ == "__main__":
    pool = Pool(cpu_count())

    nn1=2**n-1

    lt=[]
    lk=[]

    count={}


    for t in range(nn1):

        for i in range(t+1,nn1):
            lt.append([t])
            lk.append(i)

    ln_2=[nn1-1]* len(lt)
    lnn1=[nn1]* len(lt)



    print(list(zip(lt,lk,ln_2,lnn1)))


    #parallel execution of ProcessRT function
    res5 = pool.starmap_async(ProcessRT, zip(lt,lk,ln_2,lnn1))
    print(res5.get())# print the list of resulting dictionaries
    print(sum(merge(res5.get()).values()))#print the number of clos

    pool.close()  # 'TERM'
    pool.join()   # 'KILL'
```

```
): 0, (6, 5): 0, (7, 5): 0, (8, 5): 0}, {(2, 5): 0, (3, 5): 0, (
4, 5): 0, (5, 5): 0, (6, 5): 0, (7, 5): 0}, {(2, 5): 0, (3, 5):
0, (4, 5): 0, (5, 5): 0, (6, 5): 0, (7, 5): 0, (8, 5): 0}, {(2,
5): 0, (3, 5): 0, (4, 5): 0, (5, 5): 0, (6, 5): 0, (7, 5): 0}, {
(2, 5): 0, (3, 5): 0, (4, 5): 0, (5, 5): 0, (6, 5): 0}, {(2, 5):
0, (3, 5): 0, (4, 5): 0}, {(2, 5): 0, (3, 5): 0, (4, 5): 0, (5,
5): 0}, {(2, 5): 0, (3, 5): 0, (4, 5): 0, (5, 5): 0}, {(2, 5): 0
, (3, 5): 0, (4, 5): 0}, {(2, 5): 0, (3, 5): 0}, {(2, 5): 0, (3,
5): 0}, {(2, 5): 0}, {(2, 6): 0, (3, 6): 0, (4, 6): 0, (5, 6): 1
2, (6, 6): 472, (7, 6): 3156, (8, 6): 7377, (9, 6): 7665, (10, 6
): 3935, (11, 6): 1098, (12, 6): 153, (13, 6): 9}, {(2, 6): 0, (
3, 6): 0, (4, 6): 0, (5, 6): 10, (6, 6): 214, (7, 6): 945, (8, 6
): 1115, (9, 6): 382, (10, 6): 35, (11, 6): 1}, {(2, 6): 0, (3,
```

```
6): 0, (4, 6): 2, (5, 6): 78, (6, 6): 866, (7, 6): 3555, (8, 6):
6030, (9, 6): 4650, (10, 6): 1662, (11, 6): 245, (12, 6): 10}, {
(2, 6): 0, (3, 6): 0, (4, 6): 0, (5, 6): 15, (6, 6): 298, (7, 6)
: 1584, (8, 6): 3059, (9, 6): 2564, (10, 6): 1022, (11, 6): 203,
(12, 6): 22, (13, 6): 1}, {(2, 6): 0, (3, 6): 0, (4, 6): 2, (5,
6): 70, (6, 6): 680, (7, 6): 2418, (8, 6): 3427, (9, 6): 2102, (
```

In [9]:
```
sum(merge(res5.get()).values())# the computed value A334254 for n=5
```

Out[9]:
```
702525
```

In [6]:
```

```

```
%%time



from multiprocess import Pool
from multiprocess import cpu_count


n=4

#computing A334254 for n=4 by levels
if __name__ == "__main__":
    pool = Pool(cpu_count())

    nn1=2**n-1

    lt=[]
    lk=[]

    count={}


    for t in range(nn1):

        for i in range(t+1,nn1):
            lt.append([t])
            lk.append(i)

    ln_2=[nn1-1]* len(lt)
    lnn1=[nn1]* len(lt)



    print(list(zip(lt,lk,ln_2,lnn1)))


    #parallel execution of ProcessRT function
    res4 = pool.starmap_async(ProcessRT, zip(lt,lk,ln_2,lnn1))
    print(res4.get())# print the list of resulting dictionaries
    print(sum(merge(res4.get()).values()))#print the number of clos

    pool.close()  # 'TERM'
    pool.join()   # 'KILL'
```

```
[([0], 1, 14, 15), ([0], 2, 14, 15), ([0], 3, 14, 15), ([0], 4, 14
, 15), ([0], 5, 14, 15), ([0], 6, 14, 15), ([0], 7, 14, 15), ([0],
8, 14, 15), ([0], 9, 14, 15), ([0], 10, 14, 15), ([0], 11, 14, 15)
, ([0], 12, 14, 15), ([0], 13, 14, 15), ([0], 14, 14, 15), ([1], 2
, 14, 15), ([1], 3, 14, 15), ([1], 4, 14, 15), ([1], 5, 14, 15), (
[1], 6, 14, 15), ([1], 7, 14, 15), ([1], 8, 14, 15), ([1], 9, 14,
15), ([1], 10, 14, 15), ([1], 11, 14, 15), ([1], 12, 14, 15), ([1]
, 13, 14, 15), ([1], 14, 14, 15), ([2], 3, 14, 15), ([2], 4, 14, 1
5), ([2], 5, 14, 15), ([2], 6, 14, 15), ([2], 7, 14, 15), ([2], 8,
14, 15), ([2], 9, 14, 15), ([2], 10, 14, 15), ([2], 11, 14, 15), (
[2], 12, 14, 15), ([2], 13, 14, 15), ([2], 14, 14, 15), ([3], 4, 1
4, 15), ([3], 5, 14, 15), ([3], 6, 14, 15), ([3], 7, 14, 15), ([3]
, 8, 14, 15), ([3], 9, 14, 15), ([3], 10, 14, 15), ([3], 11, 14, 1
```

5), ([3], 12, 14, 15), ([3], 13, 14, 15), ([3], 14, 14, 15), ([4], 5, 14, 15), ([4], 6, 14, 15), ([4], 7, 14, 15), ([4], 8, 14, 15), ([4], 9, 14, 15), ([4], 10, 14, 15), ([4], 11, 14, 15), ([4], 12, 14, 15), ([4], 13, 14, 15), ([4], 14, 14, 15), ([5], 6, 14, 15), ([5], 7, 14, 15), ([5], 8, 14, 15), ([5], 9, 14, 15), ([5], 10, 14, 15), ([5], 11, 14, 15), ([5], 12, 14, 15), ([5], 13, 14, 15), ([5], 14, 14, 15), ([6], 7, 14, 15), ([6], 8, 14, 15), ([6], 9, 14, 15), ([6], 10, 14, 15), ([6], 11, 14, 15), ([6], 12, 14, 15), ([6], 13, 14, 15), ([6], 14, 14, 15), ([7], 8, 14, 15), ([7], 9, 14, 15), ([7], 10, 14, 15), ([7], 11, 14, 15), ([7], 12, 14, 15), ([7], 13, 14, 15), ([7], 14, 14, 15), ([8], 9, 14, 15), ([8], 10, 14, 15), ([8], 11, 14, 15), ([8], 12, 14, 15), ([8], 13, 14, 15), ([8], 14, 14, 15), ([9], 10, 14, 15), ([9], 11, 14, 15), ([9], 12, 14, 15), ([9], 13, 14, 15), ([9], 14, 14, 15), ([10], 11, 14, 15), ([10], 12, 14, 15), ([10], 13, 14, 15), ([10], 14, 14, 15), ([11], 12, 14, 15), ([11], 13, 14, 15), ([11], 14, 14, 15), ([12], 13, 14, 15), ([12], 14, 14, 15), ([13], 14, 14, 15)]
[{(2, 0): 0, (3, 0): 0, (4, 0): 0}, {(2, 0): 0, (3, 0): 0, (4, 0): 0}, {(2, 0): 0, (3, 0): 0, (4, 0): 0, (5, 0): 0}, {(2, 0): 0, (3, 0): 0, (4, 0): 0}, {(2, 0): 0, (3, 0): 0, (4, 0): 0, (5, 0): 0}, {(2, 0): 0, (3, 0): 0, (4, 0): 0, (5, 0): 0}, {(2, 0): 0, (3, 0): 0, (4, 0): 0}, {(2, 0): 0, (3, 0): 0, (4, 0): 0}, {(2, 0): 0, (3, 0): 0, (4, 0): 0, (5, 0): 0}, {(2, 0): 0, (3, 0): 0, (4, 0): 0, (5, 0): 0}, {(2, 0): 0, (3, 0): 0, (4, 0): 0}, {(2, 0): 0, (3, 0): 0}, {(2, 0): 0, (3, 0): 0}, {(2, 0): 0}, {(2, 1): 0, (3, 1): 0, (4, 1): 1, (5, 1): 25, (6, 1): 67, (7, 1): 12}, {(2, 1): 0, (3, 1): 0, (4, 1): 0, (5, 1): 8, (6, 1): 21, (7, 1): 5}, {(2, 1): 0, (3, 1): 0, (4, 1): 0, (5, 1): 5, (6, 1): 13, (7, 1): 1}, {(2, 1): 0, (3, 1): 0, (4, 1): 0, (5, 1): 4, (6, 1): 8}, {(2, 1): 0, (3, 1): 0, (4, 1): 3, (5, 1): 15, (6, 1): 8}, {(2, 1): 0, (3, 1): 0, (4, 1): 1, (5, 1): 5}, {(2, 1): 0, (3, 1): 0, (4, 1): 0}, {(2, 1): 0, (3, 1): 0, (4, 1): 0, (5, 1): 0}, {(2, 1): 0, (3, 1): 0, (4, 1): 0, (5, 1): 0}, {(2, 1): 0, (3, 1): 0, (4, 1): 0}, {(2, 1): 0, (3, 1): 0}, {(2, 1): 0}, {(2, 2): 0, (3, 2): 0, (4, 2): 0, (5, 2): 8, (6, 2): 21, (7, 2): 5}, {(2, 2): 0, (3, 2): 0, (4, 2): 0, (5, 2): 5, (6, 2): 13, (7, 2): 1}, {(2, 2): 0, (3, 2): 0, (4, 2): 3, (5, 2): 18, (6, 2): 14}, {(2, 2): 0, (3, 2): 0, (4, 2): 0, (5, 2): 1, (6, 2): 2}, {(2, 2): 0, (3, 2): 0, (4, 2): 1, (5, 2): 5}, {(2, 2): 0, (3, 2): 0, (4, 2): 0}, {(2, 2): 0, (3, 2): 0, (4, 2): 0, (5, 2): 0}, {(2, 2): 0, (3, 2): 0, (4, 2): 0, (5, 2): 0}, {(2, 2): 0, (3, 2): 0, (4, 2): 0}, {(2, 2): 0, (3, 2): 0}, {(2, 2): 0, (3, 2): 0}, {(2, 2): 0}, {(2, 3): 0, (3, 3): 0, (4, 3): 3, (5, 3): 21, (6, 3): 20}, {(2, 3): 0, (3, 3): 0, (4, 3): 4, (5, 3): 31, (6, 3): 44, (7, 3): 8}, {(2, 3): 0, (3, 3): 0, (4, 3): 3, (5, 3): 18, (6, 3): 12, (7, 3): 1}, {(2, 3): 0, (3, 3): 0, (4, 3): 0, (5, 3): 6, (6, 3): 3}, {(2, 3): 0, (3, 3): 0, (4, 3): 0}, {(2, 3): 0, (3, 3): 0, (4, 3): 0, (5, 3): 0}, {(2, 3): 0, (3, 3): 0, (4, 3): 0, (5, 3): 0}, {(2, 3): 0, (3, 3): 0, (4, 3): 0}, {(2, 3): 0, (3, 3): 0}, {(2, 3): 0}, {(2, 4): 0, (3, 4): 0, (4, 4): 0, (5, 4): 1, (6, 4): 2}, {(2, 4): 0, (3, 4): 0, (4, 4): 0, (5, 4): 1, (6, 4): 2}, {(2, 4): 0, (3, 4): 0, (4, 4): 1, (5, 4): 5}, {(2, 4): 0, (3, 4): 0, (4, 4): 0}, {(2, 4): 0, (3, 4): 0, (4, 4): 0, (5, 4): 0}, {(2, 4): 0, (3, 4): 0, (4, 4): 0, (5, 4): 0}, {(2, 4): 0, (3, 4): 0, (4, 4): 0}, {(2, 4): 0, (3, 4): 0}, {(2, 4): 0, (3, 4): 0}, {(2, 4): 0}, {(2, 5): 0, (3, 5): 0, (4, 5): 3, (5, 5): 18, (6, 5): 12, (7, 5): 1}, {(2, 5): 0, (3, 5): 0, (4, 5): 0, (5, 5): 6, (6, 5): 3}, {(2, 5): 0, (3, 5): 0, (4, 5): 0}, {(2, 5):

```
0, (3, 5): 0, (4, 5): 0, (5, 5): 0}, {(2, 5): 0, (3, 5): 0, (4, 5)
: 0, (5, 5): 0}, {(2, 5): 0, (3, 5): 0, (4, 5): 0}, {(2, 5): 0, (3
, 5): 0}, {(2, 5): 0, (3, 5): 0}, {(2, 5): 0}, {(2, 6): 0, (3, 6):
0, (4, 6): 0, (5, 6): 6, (6, 6): 3}, {(2, 6): 0, (3, 6): 0, (4, 6)
: 0}, {(2, 6): 0, (3, 6): 0, (4, 6): 0, (5, 6): 0}, {(2, 6): 0, (3
, 6): 0, (4, 6): 0, (5, 6): 0}, {(2, 6): 0, (3, 6): 0, (4, 6): 0},
{(2, 6): 0, (3, 6): 0}, {(2, 6): 0, (3, 6): 0}, {(2, 6): 0}, {(2,
7): 0, (3, 7): 0, (4, 7): 0}, {(2, 7): 0, (3, 7): 0, (4, 7): 1, (5
, 7): 5}, {(2, 7): 0, (3, 7): 0, (4, 7): 0, (5, 7): 1}, {(2, 7): 0
, (3, 7): 0, (4, 7): 1}, {(2, 7): 0, (3, 7): 0}, {(2, 7): 0, (3, 7
): 0}, {(2, 7): 0}, {(2, 8): 0, (3, 8): 0}, {(2, 8): 0, (3, 8): 0}
, {(2, 8): 0, (3, 8): 0}, {(2, 8): 0, (3, 8): 0}, {(2, 8): 0, (3,
8): 0}, {(2, 8): 0}, {(2, 9): 0, (3, 9): 0, (4, 9): 0}, {(2, 9): 0
, (3, 9): 0, (4, 9): 0}, {(2, 9): 0, (3, 9): 0}, {(2, 9): 0, (3, 9
): 0}, {(2, 9): 0}, {(2, 10): 0, (3, 10): 0, (4, 10): 0}, {(2, 10)
: 0, (3, 10): 0}, {(2, 10): 0, (3, 10): 0}, {(2, 10): 0}, {(2, 11)
: 0, (3, 11): 0}, {(2, 11): 0, (3, 11): 0}, {(2, 11): 0}, {(2, 12)
: 0}, {(2, 12): 0}, {(2, 13): 0}]
545
CPU times: user 31 ms, sys: 43.9 ms, total: 75 ms
Wall time: 86 ms
```

In [11]:
```
sum(merge(res4.get()).values())# the computed value A334254 for n=4
```

Out[11]:
```
545
```

In [8]:

```
%%time



from multiprocess import Pool
from multiprocess import cpu_count


n=3

#computing A334254 for n=3 by levels
if __name__ == "__main__":
    pool = Pool(cpu_count())

    nn1=2**n-1

    lt=[]
    lk=[]

    count={}


    for t in range(nn1):

        for i in range(t+1,nn1):
            lt.append([t])
            lk.append(i)

    ln_2=[nn1-1]* len(lt)
    lnn1=[nn1]* len(lt)



    print(list(zip(lt,lk,ln_2,lnn1)))


    #parallel execution of ProcessRT function
    res3 = pool.starmap_async(ProcessRT, zip(lt,lk,ln_2,lnn1))
    print(res3.get())# print the list of resulting dictionaries
    print(sum(merge(res3.get()).values()))#print the number of clos

    pool.close()  # 'TERM'
    pool.join()   # 'KILL'
```

```
[([0], 1, 6, 7), ([0], 2, 6, 7), ([0], 3, 6, 7), ([0], 4, 6, 7), (
[0], 5, 6, 7), ([0], 6, 6, 7), ([1], 2, 6, 7), ([1], 3, 6, 7), ([1
], 4, 6, 7), ([1], 5, 6, 7), ([1], 6, 6, 7), ([2], 3, 6, 7), ([2],
4, 6, 7), ([2], 5, 6, 7), ([2], 6, 6, 7), ([3], 4, 6, 7), ([3], 5,
6, 7), ([3], 6, 6, 7), ([4], 5, 6, 7), ([4], 6, 6, 7), ([5], 6, 6,
7)]
[{(2, 0): 0, (3, 0): 0}, {(2, 0): 0, (3, 0): 0}, {(2, 0): 0, (3, 0
): 0}, {(2, 0): 0, (3, 0): 0}, {(2, 0): 0, (3, 0): 0}, {(2, 0): 0}
, {(2, 1): 0, (3, 1): 1, (4, 1): 4}, {(2, 1): 0, (3, 1): 0, (4, 1)
: 1}, {(2, 1): 0, (3, 1): 0}, {(2, 1): 0, (3, 1): 0}, {(2, 1): 0},
{(2, 2): 0, (3, 2): 0, (4, 2): 1}, {(2, 2): 0, (3, 2): 0}, {(2, 2)
: 0, (3, 2): 0}, {(2, 2): 0}, {(2, 3): 0, (3, 3): 0}, {(2, 3): 0,
(3, 3): 1}, {(2, 3): 0}, {(2, 4): 0}, {(2, 4): 0}, {(2, 5): 0}]
```

```
8
CPU times: user 21 ms, sys: 32 ms, total: 53.1 ms
Wall time: 57 ms
```

In [10]:
```
sum(merge(res3.get()).values())# the computed value A334254 for n=3
```

Out[10]:
```
8
```