In [11]:
```python
import scipy.special as sc
import numpy as np
import math
import datetime
from tqdm import tqdm_notebook as tqdm
import pandas as pd
```

In [2]:
```python
def comb(n, k):
    return sc.comb(n, k, exact=True)

def get_hilbert_border(n, k):
    if (n == k):
        return 1
    prev_prev_d = 0
    cur_sum = comb(n - 1, prev_prev_d)
    while cur_sum + comb(n - 1, prev_prev_d + 1) < 2 ** (n - k):
        cur_sum += comb(n - 1, prev_prev_d + 1)
        prev_prev_d += 1
    return prev_prev_d + 2
```

In [3]:
```python
def get_all_vectors(sz):
    def add_elem(vecs, e):
        return list(map(lambda vec: np.append(vec, e), vecs))


    res = [np.asarray([], dtype=int)]
    for i in range(sz):
        res = add_elem(res, 0) + add_elem(res, 1)
    return list(map(lambda x: np.ndarray.flatten(x), res))

def any_from(s):
    for x in s:
        return x
```

In [4]:
```python
# Общая идея взята из доказательсва соотвествующей границы
# Берём любой доступный вектор(изначально все кроме нулевого)
# Далее поддерживаем линейный комбинации длины от 1 по d - 2,
# которые будем удалять из множества доступных
def build_hilbert(n, k):
    r = n - k
    d = get_hilbert_border(n, k)
    if (d < 3):
        raise NameError("d is less than 3")
    possible_vectors = set(map(lambda x: tuple(x), get_all_vectors(r)))
    possible_vectors.discard(tuple(np.zeros(r, dtype=int)))
    ans = []
    # Храним для каждой длины линейной комбинации все такие комбинации из уже нобранных векторов
    combs = [[] for _ in range(d - 1)]
    # Хак чтобы автоматически добавлять новый вектор в combs[1]
    combs[0].append(np.zeros(r, dtype=int))
    for i in range(n - 1):
        new_colum = np.array(any_from(possible_vectors))
        ans.append(new_colum)
        max_columns = min(d - 3, i)
        # Что мы добавим к соотв ячейкам массива comb
        additions = [[] for _ in range(max_columns + 2)]
        # Перебераем все комбинации от 0 до max_colimns векторов без нового вектора,
        # добавляя данный вектор получая новую комбинацию длины на 1 больше.
        # Таким образм мы рассмотрим лишь все новые комбинации и ничего лишнего
        for j in range(0, max_columns + 1):
            for vec in combs[j]:
                new_vec = new_colum ^ vec
                additions[j + 1].append(new_vec)
                possible_vectors.discard(tuple(new_vec))
        for j in range(1, max_columns + 2):
            combs[j].extend(additions[j])
    ans.append(np.array(any_from(possible_vectors)))
    return np.array(ans).T, d
```

In [17]:
```python
build_hilbert(14, 8)
```

Out[17]:
```
(array([[0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0],
        [1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1],
        [0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1],
        [1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1],
        [1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0],
        [0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0]]), 3)
```

In [8]:
```python
MAXN = 25
results = np.full((MAXN, MAXN), -1)
```

In [9]:
```python
params = [(n, k) for n in range(5, MAXN) for k in range(1, n - 3)]
for n, k in tqdm(params):
    try:
        start_time = datetime.datetime.now()
        H, d = build_hilbert(n, k)
        timedelta = (datetime.datetime.now() - start_time).seconds
#             print(n, k, timedelta)
        results[n][k] = timedelta
    except:
        results[n][k] = timedelta
```

In [16]:
```python
print(pd.DataFrame(results[:, :MAXN - 3]))
```

```
    0    1   2   3   4   5   6   7   8   9  ...  12  13  14  15  16  17  18  \
0  -1   -1  -1  -1  -1  -1  -1  -1  -1  -1  ...  -1  -1  -1  -1  -1  -1  -1   
1  -1   -1  -1  -1  -1  -1  -1  -1  -1  -1  ...  -1  -1  -1  -1  -1  -1  -1   
2  -1   -1  -1  -1  -1  -1  -1  -1  -1  -1  ...  -1  -1  -1  -1  -1  -1  -1   
3  -1   -1  -1  -1  -1  -1  -1  -1  -1  -1  ...  -1  -1  -1  -1  -1  -1  -1   
4  -1   -1  -1  -1  -1  -1  -1  -1  -1  -1  ...  -1  -1  -1  -1  -1  -1  -1   
5  -1    0  -1  -1  -1  -1  -1  -1  -1  -1  ...  -1  -1  -1  -1  -1  -1  -1   
6  -1    0   0  -1  -1  -1  -1  -1  -1  -1  ...  -1  -1  -1  -1  -1  -1  -1   
7  -1    0   0   0  -1  -1  -1  -1  -1  -1  ...  -1  -1  -1  -1  -1  -1  -1   
8  -1    0   0   0   0  -1  -1  -1  -1  -1  ...  -1  -1  -1  -1  -1  -1  -1   
9  -1    0   0   0   0   0  -1  -1  -1  -1  ...  -1  -1  -1  -1  -1  -1  -1   
10 -1    0   0   0   0   0   0  -1  -1  -1  ...  -1  -1  -1  -1  -1  -1  -1   
11 -1    0   0   0   0   0   0   0  -1  -1  ...  -1  -1  -1  -1  -1  -1  -1   
12 -1    0   0   0   0   0   0   0   0  -1  ...  -1  -1  -1  -1  -1  -1  -1   
13 -1    0   0   0   0   0   0   0   0   0  ...  -1  -1  -1  -1  -1  -1  -1   
14 -1    0   0   0   0   0   0   0   0   0  ...  -1  -1  -1  -1  -1  -1  -1   
15 -1    0   0   0   0   0   0   0   0   0  ...  -1  -1  -1  -1  -1  -1  -1   
16 -1    0   0   0   0   0   0   0   0   0  ...   0  -1  -1  -1  -1  -1  -1   
17 -1    0   0   0   0   0   0   0   0   0  ...   0   0  -1  -1  -1  -1  -1   
18 -1    1   0   0   0   0   0   0   0   0  ...   0   0   0  -1  -1  -1  -1   
19 -1    4   1   0   0   0   0   0   0   0  ...   0   0   0   0  -1  -1  -1   
20 -1    8   3   1   0   0   0   0   0   0  ...   0   0   0   0   0  -1  -1   
21 -1   17   8   3   1   0   0   0   0   0  ...   0   0   0   0   0   0  -1   
22 -1   38  17   8   3   1   0   0   0   0  ...   0   0   0   0   0   0   0   
23 -1   78  38  17   8   3   2   1   0   0  ...   0   0   0   0   0   0   0   
24 -1  180  75  38  18   8   3   1   0   0  ...   0   0   0   0   0   0   0   

    19  20  21  
0   -1  -1  -1  
1   -1  -1  -1  
2   -1  -1  -1  
3   -1  -1  -1  
4   -1  -1  -1  
5   -1  -1  -1  
6   -1  -1  -1  
7   -1  -1  -1  
8   -1  -1  -1  
9   -1  -1  -1  
10  -1  -1  -1  
11  -1  -1  -1  
12  -1  -1  -1  
13  -1  -1  -1  
14  -1  -1  -1  
15  -1  -1  -1  
16  -1  -1  -1  
17  -1  -1  -1  
18  -1  -1  -1  
19  -1  -1  -1  
20  -1  -1  -1  
21  -1  -1  -1  
22  -1  -1  -1  
23   0  -1  -1  
24   0   0  -1  

[25 rows x 22 columns]
```

Как видно в переделах одного n время работы падает с ростом k. Аналогично же при фиксированном k с ростом n растёт время.

Это можно объяснить оценив ассимптотику работы. Заметим, что основную сложность алгоритма состовляет пресчёт массива $comb$. На после шага с номерером i(нумерация с 1) каждая ячейка его содержит все возможные C_i^t комбинаций, где $t = min(d - 2, i)$. При этом каждый последующий шаг "трогает" все вектора с предыдущего. Пересчёт же каждого происходит за $r = n - k$

В таком случаем можно оценить время работы как $\sum\limits_{i=1}^{n-1} \sum\limits_{j=0}^{\min(d-2, i-1)} C_{i-1}^{j} \cdot r$

Т.к. с ростом k уменьшается d это объясняет ускорение алгоритма, ибо от уменьшения r данная сумма уменьшается не так сильно.

Отсюда можно сделать вывод, что для k близких к n код отрабатывает за очень даже разумное время, однако при скорости кода менее $\frac{1}{2}$ при $n \geq$ уже придётся ждать несколько минут.

In [ ]:

In [ ]: