属性1　属性2　属性3
2　　3　　3
假设空间中有 3 * 4 * 4 + 1 = 49种假设。
在不考虑沉余的情况下，最多包含 k 个合取式来表达假设空间，显然 k 的最大值是49。
但是其中包含了很多冗余的情况。

若考虑冗余的情况（忽略空集）：
48种假设中：
（1）具体假设：2 * 3 * 3 = 18 种
（2）1个属性泛化： 3 * 3 + 2 * 3 + 2 * 3 = 21种
　　|　　|　　|
　　1泛化　2泛化　3泛化
（3）2个属性泛化：2 + 3 + 3 = 8 种
（4）3个属性泛化：1 种
若考虑冗余，k 最大取值为18.（若大于18，则必含有泛化假设，泛化会减少具体假设数目，假设数目将小于18，与大于18矛盾）
这时需要根据 k 取值的不同分清况讨论。

（1）k = 1时，任选一种假设都可以作为一种没有沉余的假设，共48种。
（2）k = 18时，就是18种具体属性假设的析取式，共1种。
（3）1 < k < 18时，需要另加分析。

算法：
由于属性泛化后，一个泛化的假设可以对应多个具体假设。
把所有假设按三属性泛化，二属性泛化，一属性泛化，具体属性排序
(这样可以保证排在后面的假设不会包含前面的任何一个假设，所以省略了一些包含判断)，进行循环枚举，按顺序遍历所有假设组合2^48种可能(当然绝大部分都提前结束了，不会是那么夸张的量级，虽然也不低)：
·使用栈来实现非递归，如果当前假设还有没被析合式所包含的具体假设，则认为可以入栈，并且当前栈大小的长度计数加111，并继续扫描。
·如果当前扫描已经到了最后一个假设，或者所有具体假设已经被全部包含，则退栈。
·循环结束条件：当最后一个假设作为第一个压入栈的元素时，认为已经遍历结束。

细节设计：
a. 每个假设的表示：
每个假设对应一个32位整型(假设变量为 hypo_const),代表着它所对应了哪些具体假设，如果它包含了某种具体假设，则该位为1。
eg. 假设1： 00 0000 0000 0000 0001

b. 析合式包含的假设的表示：
由于一共有18种具体假设，可以用一个32位整型(变量为 hypos_cur)的后18位来表示每一个具体假设。用1表示具体假设没被包含，用0表示具体假设已经被析合式包含。
初始的析合式为空，可以设初试值为0X3FFFF。

c. 判断析合式是否包含了全部的具体假设：
　hypos_cur=0

d. 判断该假设是否已经被析合范式包含:

hypo_tmp = hypos_cur & hypo_const

若 hypo_tmp = 1, 入栈,
若 hypo_tmp = 0, 不入栈。

hypos_cur	hypo_const	hypo_tmp	
0	0	0	-> 析合式已包含 -> 不入栈
0	1	0	-> 析合式已包含 -> 不入栈
1	0	0	-> 析合式未包含，假设未包含 -> 不入栈
1	1	1	-> 析合式未包含，假设包含 -> 入栈

e. 入栈的操作：

hypos_cur ^= hypo_tmp

当某个假设加入析合范式后(入栈)用 hypos_cur 与 hypo_tmp 做异或运算，来更改析合式所包含的具体假设。
若可以入栈，则假设对应位 hypo_tmp = 1, hypos_cur = 1, 异或运算后为0，表示析合式包含改假设。
采用异或运算的好处是，在出栈时，再次异或即可还原 hypos_cur 原状态。

f. 出栈的操作：

hypos_cur ^= hypo_tmp

出栈时再次用 hypos_cur 与 hypo_tmp 做异或，回到加入该假设前的情况。

代码实现：

```
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <vector>
#include <stack>
using namespace std;

// 0表示*泛化情况
// 1 表示该属性下取第1种属性值，2，3同理
// 每三个一组，如000：***；123：三种属性分别取第1，2，3个属性值
// 一组中有几个0表示几属性泛化

static const char list[] = {
    0,0,0,
    0,0,1,0,0,2,0,0,3,0,1,0,0,2,0,0,3,0,1,0,0,2,0,0,
    0,1,1,0,1,2,0,1,3,0,2,1,0,2,2,0,2,3,0,3,1,0,3,2,0,3,3,
    1,0,1,1,0,2,1,0,3,2,0,1,2,0,2,2,0,3,
    1,1,0,1,2,0,1,3,0,2,1,0,2,2,0,2,3,0,
    1,1,1,1,1,2,1,1,3,1,2,1,1,2,2,1,2,3,1,3,1,1,3,2,1,3,3,
    2,1,1,2,1,2,2,1,3,2,2,1,2,2,2,2,2,3,2,3,1,2,3,2,2,3,3
};

class hypos {
public:
        virtual int insert(int cur) = 0;  //基类中的纯虚函数
};

//单个的假设类
// hypo_const 表示具体假设
class hypo: public hypos
{
public:
        hypo(int a, int b, int c)
        {
                hypo_const = 0;
                vector<char> p[3];
                // a = 0 表示第一种取泛化属性，需要把其包含的1，2两种具体属性都存入 p 容
器中
                if(a == 0)
                {
                        p[0].push_back(1);
                        p[0].push_back(2);
                }
                else p[0].push_back(a);

                if(b == 0)
```

```
            {
                p[1].push_back(1);
                p[1].push_back(2);
                p[1].push_back(3);
            }
            else p[1].push_back(b);

            if(c == 0)
            {
                p[2].push_back(1);
                p[2].push_back(2);
                p[2].push_back(3);
            }
            else p[2].push_back(c);

            for(unsigned int i = 0; i < p[0].size();i++)
                for(unsigned int j = 0; j < p[1].size(); j++)
                    for(unsigned int k = 0; k < p[2].size(); k++)
                        // 最小 1 1 1：1*9+1*3+1 = 13
                        // 这里 -13为保证右移计数从0开始，每一种假设对应一位
                        // |= 表示按位或
                        // 对于每一种具体假设，基本只用一次 如 1 1 1 -> 0...01
                        // 对于有泛化的假设，则会用到如 1 1 0
                        // 其 p[2]位有1，2，3三种属性，需要将这三种假设对应位置为1
                        // 0...0 0001, 0...0 0010, 0...0 0100 按位或 = 0... 0 0111
                        hypo_const |= (1 << (p[0][i] * 9 + p[1][j] * 3 + p[2][k]) – 13);

        }

        int insert(int cur)
        {
            // 若 hypo_const & cur = 1，则可以入栈
            return (hypo_const & cur);
        }

private:
        int hypo_const;  //表示具体假设
};

// 用于压入栈的派生类 用来实现非递归
// hypo_tmp 记录这个假设入栈时，带入了哪些具体假设，出栈时要还原
// ptr 记录入栈时的位置

class hypo_ss: public hypos
{
public:
        hypo_ss(int _ptr, int tmp)
        {
            hypo_tmp = tmp;
            ptr = _ptr;
```

```
        }

        int insert(int cur)
        {   return 0;    }

        int hypo_tmp;
        int ptr;
};

// 用来循环遍历的类
// sum 各个长度的析合式各有多少种可能
// ss 用来实现非递归的栈
// hypos_cur 当前没被包含的具体假设 初始值为0X3FFFF
// hyposs 48个假设集合

class Traversal : public hypos
{
public:
        Traversal()
        {
                hypos_cur = 0x3ffff;
                for(int i = 0; i < 48; ++i)
                {
                        //  见 list[] : (0,0,0), (0,0,1) …
                        // hypo(a, b, c) 表某个假设（属性分别为 a,b,c）
                        hyposs.push_back(hypo(list[3 * i], list[3 * i + 1], list[3 * i + 2]));
                }
        }

        //循环顺序遍历的主体
        //cur 初始的位置 设为0
        int insert(int cur)
        {
                int ptr = cur;    //当前指向的位置
                while(1)
                {
                        //退出条件 当最后一个假设作为第一个入栈的元素 表示遍历完成
                        if(ptr > 47 && !ss.size()) break;

                        //回退条件 扫描到最后或者所有具体假设都被包含
                        if(hypos_cur == 0 || ptr > 47)
                        {
                                hypo_ss hypo_tmp = ss.top();
                                hypos_cur ^= hypo_tmp.hypo_tmp;    //出栈异或
                                ptr = hypo_tmp.ptr + 1;
                                ss.pop();
                                continue;
                        }

                        //入栈条件 如果该假设还有未被包含的具体假设 则入栈，
                        // 并当前栈大小的计数加1
                        if(int tmp = hyposs[ptr].insert(hypos_cur))
```

```
                {
                        hypos_cur ^= tmp;
                        ss.push(hypo_ss(ptr, tmp));
                        if(sum.size() < ss.size())
                                sum.push_back(0);
                        sum[ss.size() - 1]++;
                }
                ptr++;
        }
        return 1;
    }
    //输出各个长度的可能数
    void print()
    {
        for(unsigned int i = 0; i < sum.size(); ++i)
            printf("length %d : %d\n", i + 1, sum[i]);
    }

private:
    vector<int> sum;
    stack<hypo_ss> ss;
    int hypos_cur;
    vector<hypo> hyposs;
};

int main()
{
    Traversal traversal;
    traversal.insert(0);
    traversal.print();
    system("pause");
    return 0;
}

/* Output:
length 1 : 48
length 2 : 931
length 3 : 10332
length 4 : 72358
length 5 : 342057
length 6 : 1141603
length 7 : 2773332
length 8 : 4971915
length 9 : 6543060
length 10 : 6175660
length 11 : 4003914
length 12 : 1676233
length 13 : 422676
length 14 : 61884
length 15 : 5346
length 16 : 435
length 17 : 27
```

length 18 : 1
sh: 1: pause: not found
*/