搜索及优化

王乃广 **2019.7.26**

经典问题八皇后伪代码

```
void dfs(int dep){
   if (dep==n+1) { 输出方案 return;}
   for (int j=1;j<=n;j++)
        if 第dep个可以放置在第dep行第j列 {
           vis[dep][j]=1;
           //记录第dep行第j列放置了一个一个皇后
           dfs(dep+1);
           vis[dep][j]=0; 消除影响
        }
}
```

例1：给出两个素数p1,p2，x满足如下条件：

(1).x以p1结尾；

(2).x是p2的倍数；

(3).x是满足条件(1),(2)的所有数中的最小值；

给出t组p1,p2，计算所有x之和。

例：p1 = 19, p2 = 23，则x = 1219。

范围：

5 < p1,p2 < 1,000,000;

t < 100,000.

算法一：朴素的枚举（时间复杂度不明确）

```
1  long long calc(int p1, int p2){
2      int t = p1; cnt = 0;
3      while(t){
4          t /= 10; cnt++;
5      }
6      //朴素枚举倍数
7      for(long long x = p2; ; x += p2){
8          if(x % mi[cnt] == p1)
9              return x;
10     }
11 }
```

算法二：稍加改进

x = p2 * k，显然：

x % 10 = p1 % 10 = (p2 % 10) * (k % 10) % 10

由此推算**k**的个位数字**y**，

接下来只需枚举个位数字是**y**的**k**。

```
for(;; y += 10){
    if(y * p2 % mi[cnt] == p1)
        return y * p2;
}
```

算法三：继续改进

在算法二确定倍数k的个位数字的基础上，

x % 100 = p2 % 100 = (p1 % 100) * (k % 100) % 100

由此可以求得k的十位数字，……

```
void dfs(int step, long long cur){
    if(ok) return;
    if(step == cnt){
        ok = true;tmp = cur * p2; return;
    }
    for(int x = 0; x < 10; x++){
        if((cur + x * mi[step]) * p2 % mi[step + 1]
            == p1 % mi[step + 1])
            dfs(step + 1, k, cur + x * mi[step]);
    }
}
```

算法三：继续改进

主程序调用：**dfs(1，y);**

极限数据下的运行时间对比：

490S　　50S　　0.1S

bfs的基本框架

```
初始状态入队列;
while( 队列非空 ){
        队首元素cur出队;
        if( cur是目标状态 ) return;
        for( int  i = 1; i <= 规则数; i++ ){
                由cur产生新状态nxt;
                if( nxt合法且没出现过 ) nxt入队列;
        }
}
```

例2:倍数(Multiple), ZOJ1136, POJ1465

题目描述：

编写程序，实现：给定一个自然数 N， N 的范围为[0, 4999]，以及 M 个不同的十进制数字 X1, X2, …, XM （至少一个，即 M≥1），求 N 的最小的正整数倍数，满足：N 的每位数字均为 X1, X2, …,XM 中的一个。

输入描述：

输入文件包含多个测试数据，测试数据之间用空行隔开。每个测试数据的格式为：第 1 行为自然数 N；第 2 行为正整数 M；接下来有 M 行，每行为一个十进制数字，分别为 X1, X2, …, XM。

输出描述：

对输入文件中的每个测试数据，输出符合条件的 N 的倍数；如果不存在这样的倍数，则输出0。

分析：

把给定的数字按升序排。例：1，2，3，可以组成的数(按从小到大的顺序)

1，2，3

11，12，13，21，22，23，

31，32，33

111，112，113

121，122，123

131，132，133

211，212，213，……

显然具有队列的特点。

输入：	输出：
22 3 7 0 1 2 1 1	110 0

考虑用BFS来按顺序生成正整数，直到能被N整除。

状态的表示：

如果队列中存生成的数，那么对于无解的情况，没有明显的结束条件。

如样例2：n=2，给定的数字只有一个1，可以生成的正整数是1,11,111,……，不存在能被2整除的正整数。

改进：

由于0<=n<=4999，生成的正整数%n不超过5000。

生成的正整数中，如果存在a%n==b%n且a<b，那么没有必要在b的基础上继续搜索。(?)

可以在队列中存余数，这样的总状态数不超过5000.

```
struct node{
        int d;          //数字
        int pre;        //前驱
        int yu;         //余数
};
```

```
memset(v,0,sizeof(v));
int head=0,tail=0;
for(int i=0;i<m;i++){
        //最高位不能是0
        if(x[i]==0) continue;
        cur.d=x[i];
        cur.pre=-1;
        cur.yu=x[i]%n;
        q[tail]=cur; tail++;
}
```

```
while(head<tail){
    cur=q[head];
    if(cur.yu==0){   //找到了
        out(head); printf("\n"); return;
    }
    for(int i=0;i<m;i++){
        nxt.d=x[i]; nxt.pre=head;
        nxt.yu=(10*cur.yu+x[i])%n;
        if(!v[nxt.yu]){
            q[tail]=nxt; tail++;
            v[nxt.yu]=true;
        }
    }
    head++;
}
```

例3:luogu.org/P2730魔板

一张有8个大小相同的格子的魔板：

1 2 3 4

8 7 6 5

魔板的每一个方格都有一种颜色。这8种颜色用前8个正整数来表示。可以用颜色的序列来表示一种魔板状态，规定从魔板的左上角开始，沿顺时针方向依次取出整数，构成一个颜色序列。对于上图的魔板状态，我们用序列（1，2，3，4，5，6，7，8）来表示。这是基本状态。

例3:luogu.org/P2730魔板

这里提供三种基本操作，分别用大写字母“A”，“B”，“C”来表示（可以通过这些操作改变魔板的状态）：

“A”：交换上下两行；

“B”：将最右边的一列插入最左边；

“C”：魔板中央四格作顺时针旋转。

例3:luogu.org/P2730魔板

对于每种可能的状态，这三种基本操作都可以使用。

你要编程计算用最少的基本操作完成基本状态到目标状态的转换，输出基本操作序列。

输入格式：

只有一行，包括8个整数，用空格分开（这些整数在范围 1——8 之间）不换行，表示目标状态。

输出格式：

Line 1: 包括一个整数，表示最短操作序列的长度。

Line 2: 在字典序中最早出现的操作序列，用字符串表示，除最后一行外，每行输出60个字符。

例3:luogu.org/P2730魔板

输入样例#1：

2 6 8 4 5 7 3 1

输出样例#1：

7

BCABCCB

说明：题目翻译来自NOCOW。USACO Training Section 3.2

算法一：bfs

状态总数以3倍递增：1，3，3^2，3^3，……，3^ans

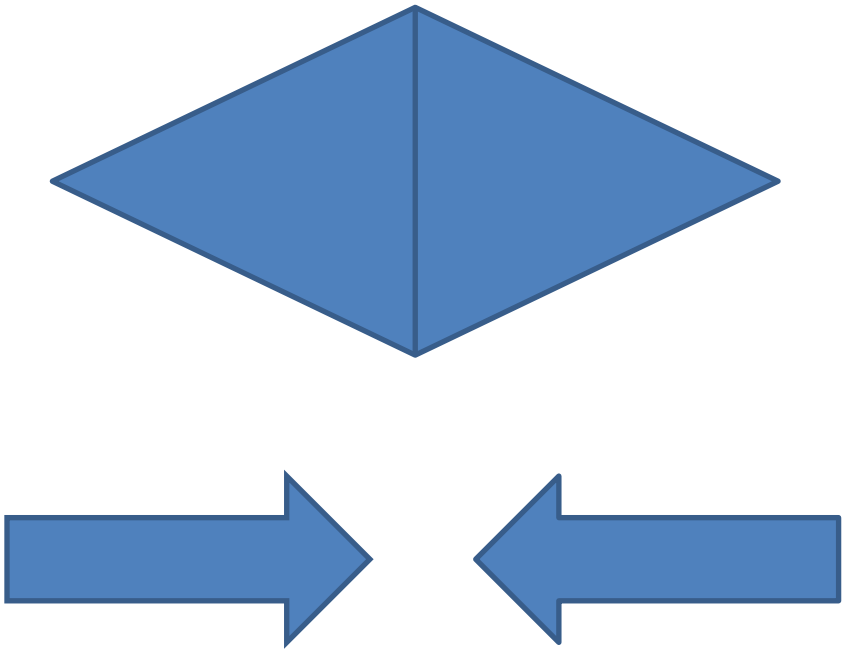
算法二：双向bfs

使用条件：题目中的规则是可逆的。

正反两个方向的bfs交替进行，

直到某方向中出现的状态

在另一个方向中出现为止。



```
while(!flag){
    while(!flag
        && c1 < sv1.size()
        && sv1[c1].step == len){
        bfs1(c1); c1++;
    }
    while(!flag
    && c2 < sv2.size()
    && sv2[c2].step == len){
        bfs2(c2); c2++;
    }
    len++;
}
```

例3:luogu.org/P2730魔板

结束标记flag的维护：

正向bfs第len层的新状态在反向bfs中已经出现，则一定出现在第len-1层，总步数：2 * len + 1;

反向bfs第len层的新状态在正向bfs中已经出现，则一定出现在第len层，总步数：2 * (len + 1)

例3:luogu.org/P2730魔板

状态的维护：

(1)每个状态用一个int s[8]来表示

(2)每个状态对应1..8的一个全排列，利用序号[0, 8!)来表示状态：

0:12345678

1:12345687

2:12345768

4:12345786

s[0,8] 与序号[0, 8!)之间互转。（康拓展开 ）

例3:luogu.org/P2730魔板

单向bfs中，只要安装A→C的顺序来扩展新状态，求得的操作序列一定字典序最早

双向bfs中，正向bfs中的操作序列没有问题，而反向bfs中不一定。

解决方法：

反向bfs中找到在正向bfs中出现过的状态后，不立刻停止，而是继续扩展，找出所有的序列方案。

最后对所有的方案进行字典序排序。

```
while(!flag){
        while(c1 < sv1.size()
        && sv1[c1].step == len){
            bfs1(c1); c1++;
        }
        if(flag) break;
        while(c2 < sv2.size()
        && sv2[c2].step == len){
            bfs2(c2); c2++;
        }
        len++;
}
```

例4：素数集合

1到9的9个数字，每个数字用且仅用1次，组成若干个素数构成的集合。{2,5,47,89,631}

这样的集合有多少个？

例4：素数集合

(1) 9个数字组成几个数？cnt = [1, 10)

(2) 9个数字组成cnt个数，这些数的位数是非递减的

找出所有符合要求的方案

(3) cnt个数的位数分别为c[0]..c[cnt-1]

把1..9填入相应的位置，使得d[0] < d[1] < … < d[cnt-1]，且均为素数。

搜索的优化：剪枝

可行性剪枝就是在这个不可能成为合法解的时候剪枝

最优性剪枝就是在这个不可能成为最优解的时候剪枝

例5：计算x^n

x^15至少需要进行5次运算。

$$n \times n \times \ldots \times n = n^{15}$$

$$n \times n = n^2$$
$$n^2 \times n^2 = n^4$$
$$n^4 \times n^4 = n^8$$
$$n^8 \times n^4 = n^{12}$$
$$n^{12} \times n^2 = n^{14}$$
$$n^{14} \times n = n^{15}$$

$$n \times n = n^2$$
$$n^2 \times n = n^3$$
$$n^3 \times n^3 = n^6$$
$$n^6 \times n^6 = n^{12}$$
$$n^{12} \times n^3 = n^{15}$$

```
int main(){
    for(int i = 1; i <= 200; i++)
        c[i] = i - 1;
    a[0] = 1; a[1] = 2;
    ans = 1;
    for(int i = 3; i <= 200; i++){
        n = i; dfs(2); ans += c[i];
    }
    printf("ans = %d\n", ans);
    return 0;
}
```

```
void dfs(int step){
    if(a[step - 1] == n){
        if(c[n] > step - 1)
            c[n] = step - 1;
    }else{
        for(int i = step - 1; i >= 0; i--){
            if(a[i] + a[step - 1] > n) continue;
            a[step] = a[i] + a[step - 1];
            dfs(step + 1);
        }
    }
}
```

```
void dfs(int step){
    //剪枝效果明显
    if(step > c[n]) return;
    if(a[step - 1] == n){
        if(c[n] > step - 1)
            c[n] = step - 1;
    }else{
        for(int i = step - 1; i >= 0; i--){
            if(a[i] + a[step - 1] > n) continue;
            a[step] = a[i] + a[step - 1];
            dfs(step + 1);
        }
    }
}
```

例6:服务站uva10160

一家公司在N(3<=N<=35)个城市销售个人电脑，城市编号为1、2、……、N。城市间有M条直通航线。公司决定在一些城市设立服务站，使得对于任意城市X，要么是服务站，要么它与服务站有直通航线相连。

编写程序，计算公司最少要设立几个服务站。

输入格式：

包含多组测试数据(不超过10组)，对于每组测试数据：

第一行，两个整数N、M，分别表示城市个数及直通航线的数量；

以下M行，每行两个整数，表示一条直通航线连接的城市编号；

0 0表示输入的结束。

输出格式：

对于每组测试数据，输出一行，表示服务站的个数。

样例输入：	样例输出：
8 12 1 2 1 6 1 8 2 3 2 6 3 4 3 5 4 5 4 7 5 6 6 7 6 8 0 0	2

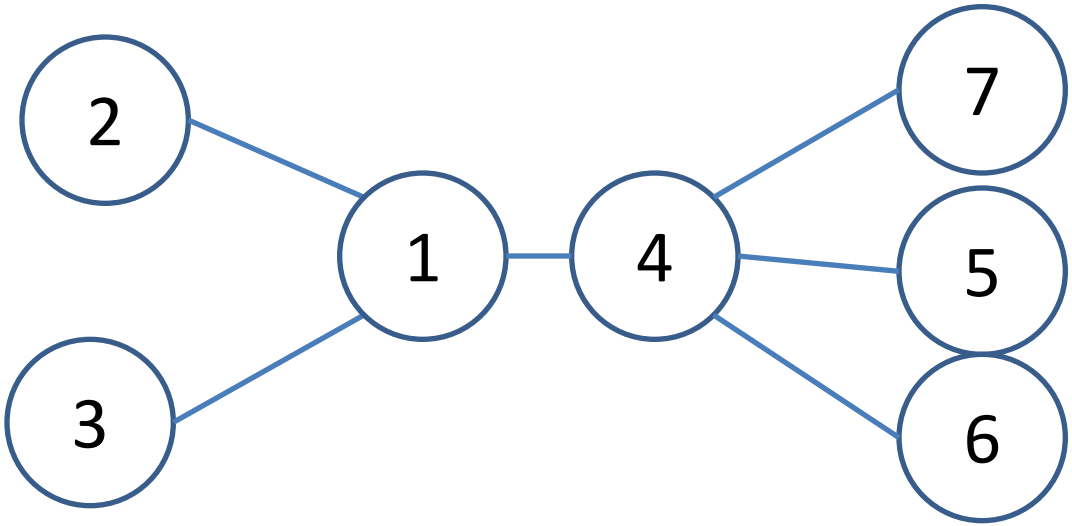
朴素的思路，每个城市要么设为服务站，要么不设为服务站，2^N。超时

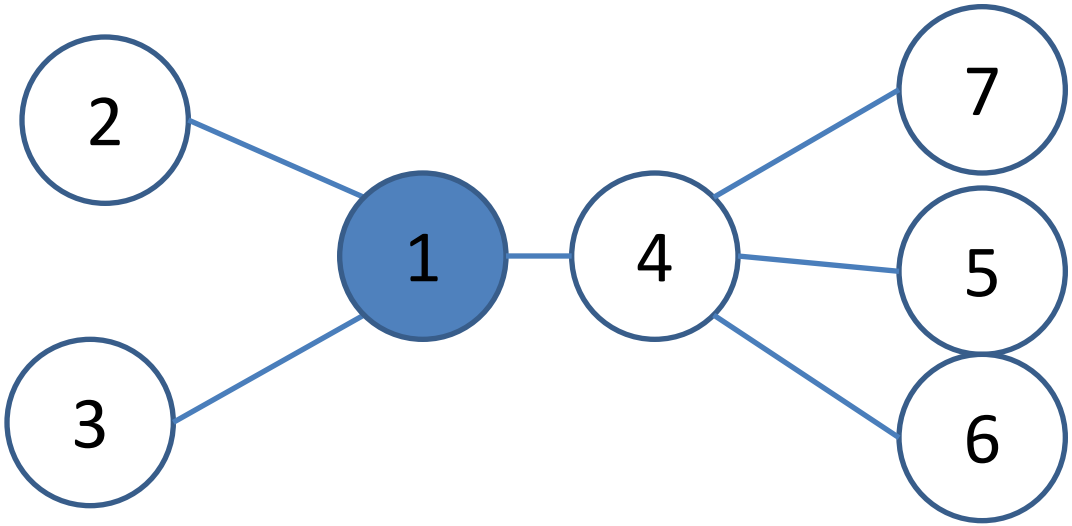
优化和剪枝：

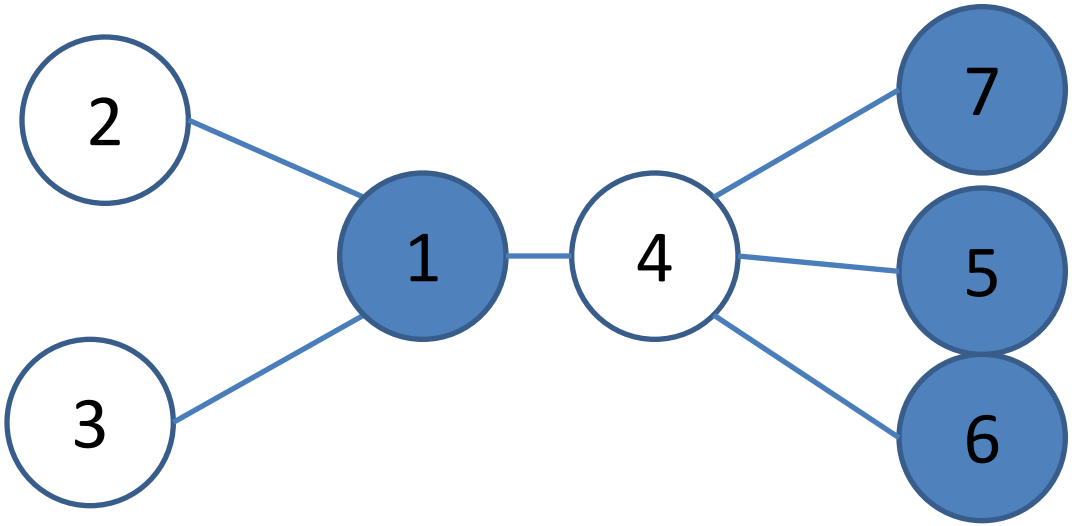
在按城市编号依次搜索的过程中，

已被覆盖的城市，不再考虑设为服务站。

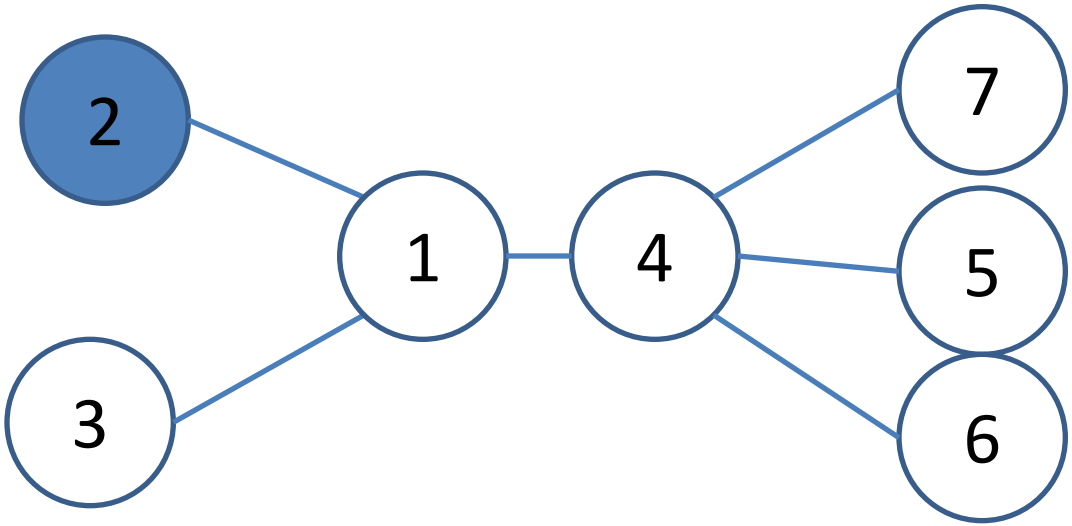
实际是错误的， 反例：

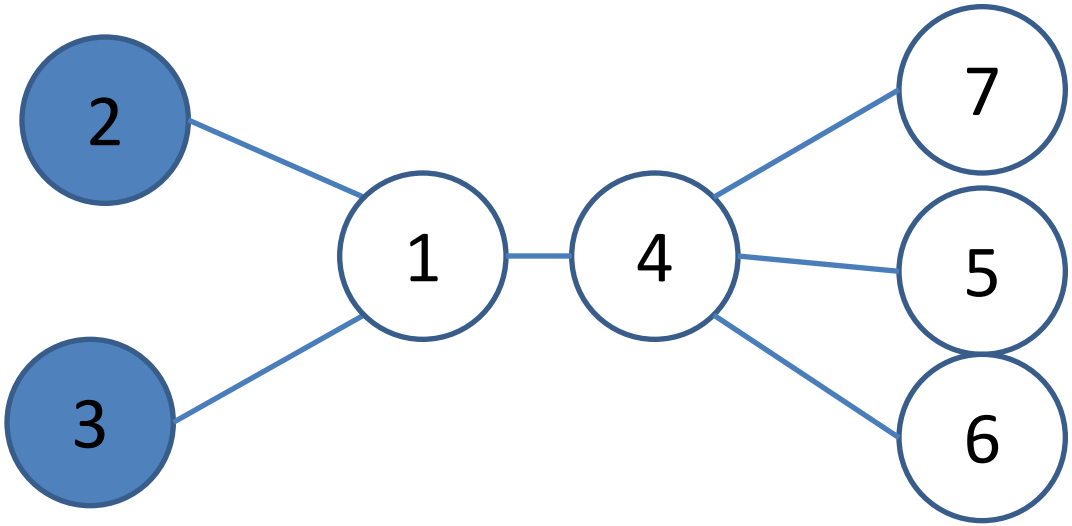

2
3
1
4
7
5
6

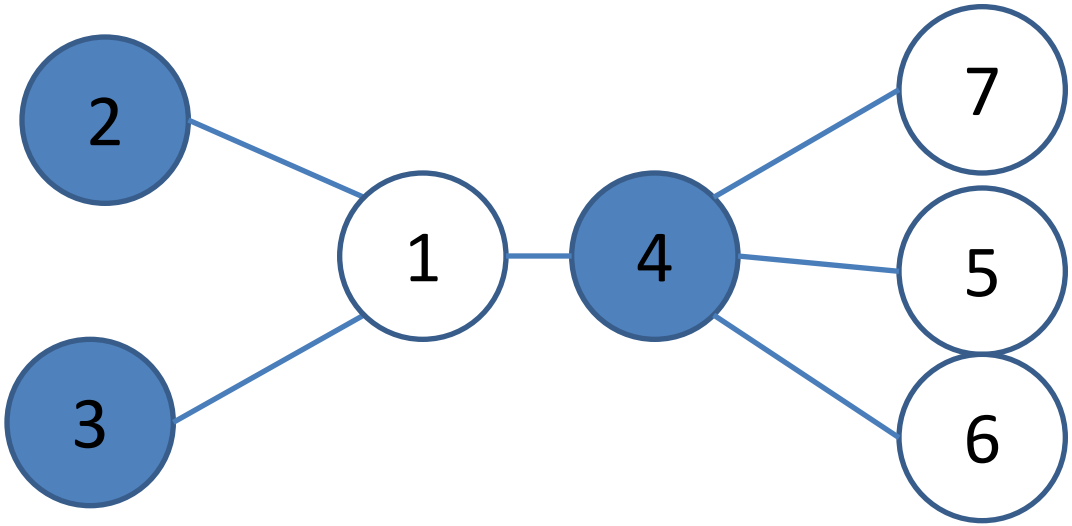

2
3
1
4
7
5
6



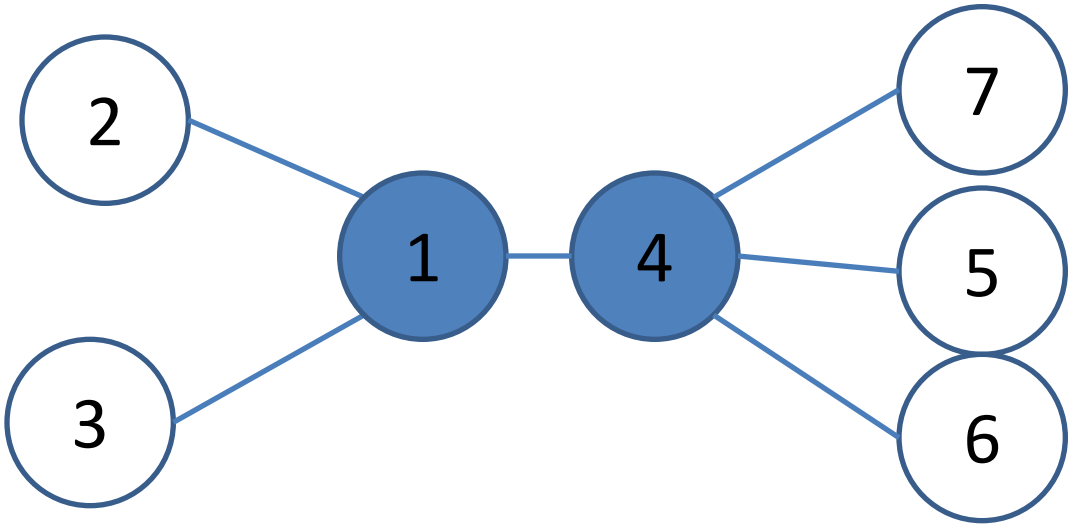
ans=4


2
7
1
4
5
3
6


2
3
1
4
7
5
6



ans=3



正确结果：ans=2。

修正：

如果把当前城市设为服务站，不会使得被覆盖的城市增加，那么该城市不需要设为服务站。

若搜索过程中设立的服务站数量已经超过当前最优值，没必要再继续下去；

对于当前城市之前的某个城市，若它没有被覆盖，且它的最大邻接点编号小于当前城市，也就是说，后面再怎么设服务站也没法覆盖这个城市，没必要再继续下去。