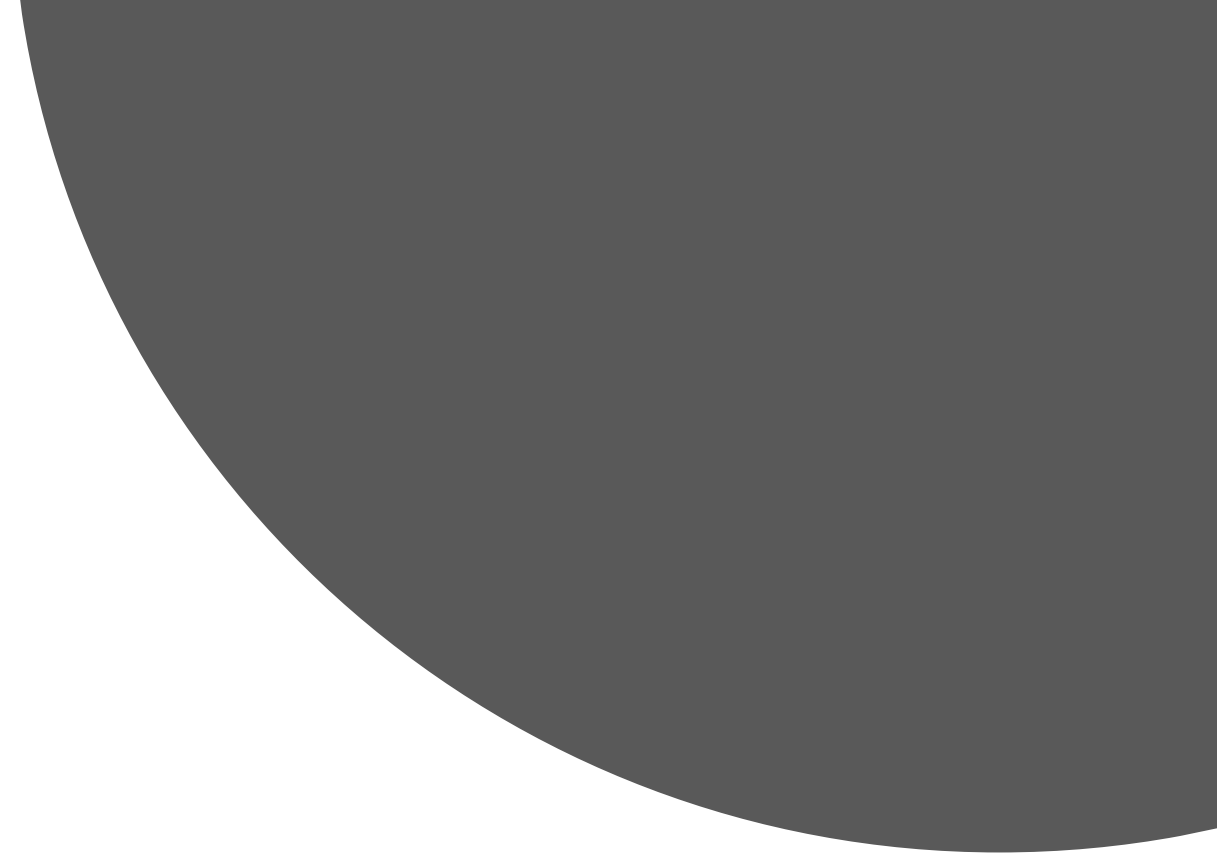
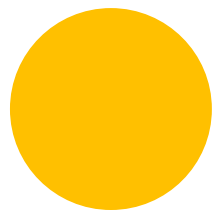
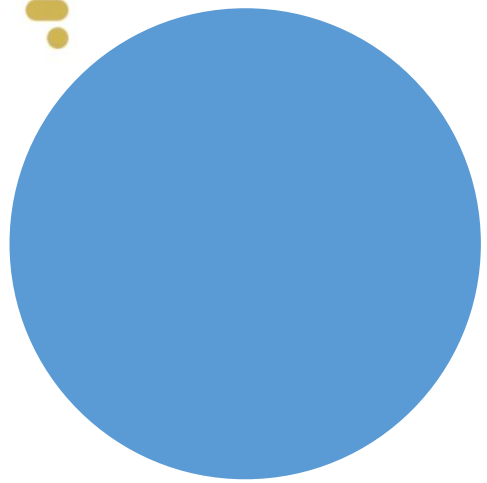




# 马上开始

35tang-C++竞赛系列四阶课程



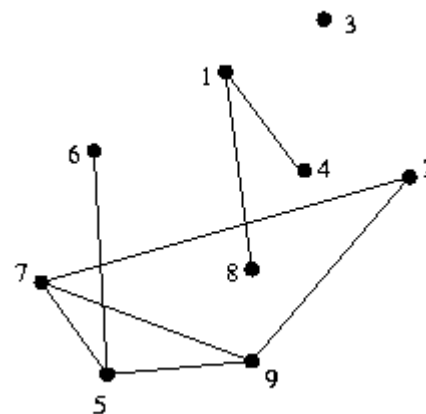
# 《 35tang-C++竞赛系列四阶课程 》




# 复习图的联通搜索：Flood Fill

BFS

DFS



右侧图10个节点构成了3个联通图，分别包含若干个节点。可以定义 `components[11]` 来保存 component numbers 单元号，比如 `components[1]` 表示1号节点所在的单元号，一共编号1到3。这里的flood fill就是要填写1到3给 `components` 数组。



DFS一般就是从一个顶点开始，找和他有连接的相邻的点，然后递归找这些邻居节点，一直到找到或者找完位置，一次找到的都是一个component的。

BFS就是从一个顶点开始，不断地把他的邻居节点加入队列，进行处理，处理后把处理过的邻居节点再加入队列，直到队列为空。直到队列为空，找到的都是一个component的。

# 并查集

并查集顾名思义就是有“合并集合”和“查找集合中的元素”两种操作的关于数据结构的一种算法。

图中连通的节点可以认为在1个集合，他们最后指向的根是一个

无向图的连通分量个数，初始时，将无向图中各个顶点视为不连通的子集合，对图中每一条边，相当于union这条边对应的两个顶点分别所在的集合，直至所有的边都处理完后，还剩下的集合的个数即为连通分量的个数（有几个parent指向自己的根元素就有几个单元）。

```
int n,m;  
int parent[1001];  
int findroot(int x)  
{  
    if (parent[x]==x) return x;  
    return (findroot(parent[x]));  
    //递归查找集合的代表元素也就是根元素  
}
```



```
for(int i=1;i<=n;i++)  parent[i]=i; //初始化n个集合
```

```
for (int i=1;i<=m;i++)    { //union的过程
```

```
    int a,b;  cin>>a>>b; //读入两个节点
```

```
    int x=findroot(a),y=findroot(b); //找到ab所在集合的根元素
```

```
    parent[x]=y; //合并两个根所在集合
```

```
}
```

```
int cnt=0;
```

```
for(int i=1;i<=n;++i){ //统计集合个数，即为连通分量个数
```

```
    if(parent[i]==i)  ++cnt;
```

```
}
```

# 2019 NOIP零件加工

---



先看题，理解题目



用测试数据验证你是否真的理解了

<https://www.luogu.com.cn/problem/P5663>



某些工人之间存在双向的零件传送带。保证每两名工人之间最多只存在一条传送带。图。

我们很容易发现，如果问题是节点a生产L，节点1是否生产原材料实际上就是从a号节点出发，能不能走L步走到节点1，也就是从节点1走L步走到a节点。

5 5 5

1 2

2 3

3 4

4 5

1 5

1 1 No

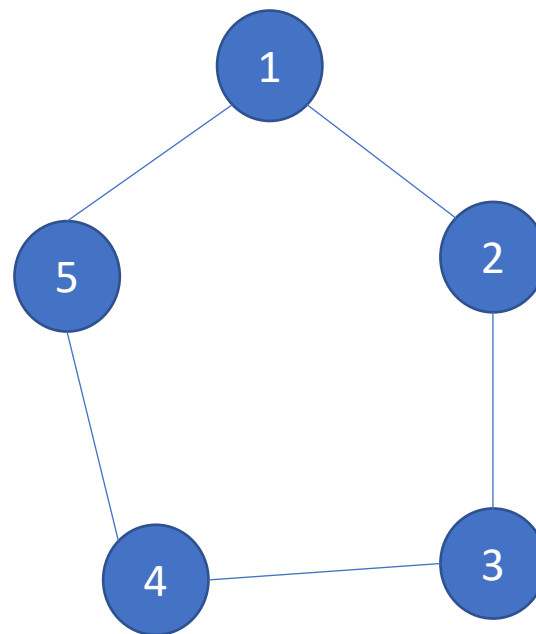
1 2 Yes

1 3 No

1 4 Yes

1 5 Yes

# 分析示例数据



# 找规律

如果还是刚才的数据，换成另外一个节点，比如问：

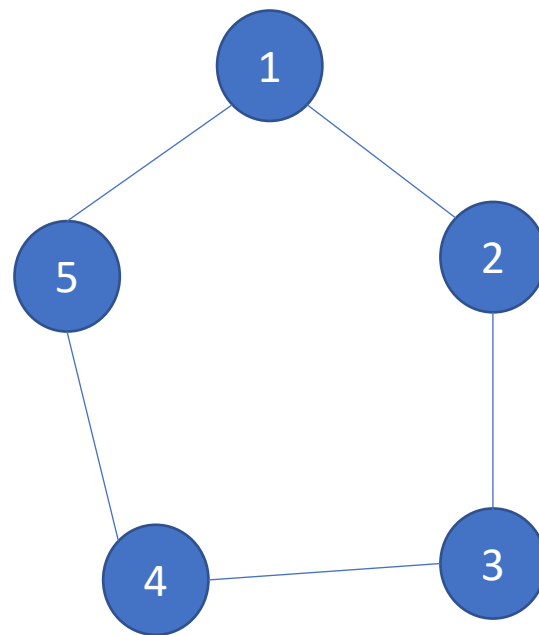
节点2生产5，生产7，生产9

节点2生产6，生产8，生产10

节点3生产4

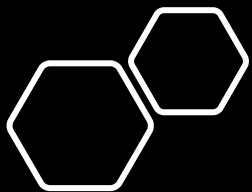
节点3生产5

节点4生产3 5 6 10



# 尝试思考

- 如果每一个都搜索?就是从节点1出发搜索, 每一次找到一个相邻节点就步数+1, 一直到到达a的时候步数是L, 问题是一个节点可能访问多次, 比如第一个到a是2步, 那么显然4步, 6步...都能再次到达a, 这样搜索重复太多了。如果2步可以到a, 实际上4步, 6步, 8步都不用搜索了, 肯定也能到? 如果发现3步可以到a呢? 那么后面5步, 7步..都不用搜索了, 肯定也能到?
- 所以: 给出一个节点a生产L, 我们要找节点1是否需要生产实际上是在找这个节点a是否可以经过L步到达节点1, 或者从节点1走L步能否到达节点a。进一步, 无论L多大, 其实只由距离的奇偶性决定, 也就是说如果L是偶数, 并且大于等于节点1到节点a最小偶数距离, 就一定可以到达1点; 奇数一样。因为在最短路径的基础上, 最后可以在两个点之间往返无数次, 每次往返步数+2, 不会改变奇偶性。比如节点1到节点a可以最短3步走到, L是11步, 那么可以3步到a, 然后在a和他的前一个节点来回走4次。
- 这么大的数据量只能 $O(N)$ 或者说遍历一次搞定



# 寻找所有节点 到节点1的最 短路径算法

注意：由于两个节点之间的距离都是1，所以可以用BFS来找最短路径，因为最先访问到的就是和1相邻的层次最少的，也就是和1距离的点数最少的，也就是最短路径。如果两个节点之间的距离不都是1，不能直接用BFS。

用一个数组保存节点1到每一个点的最短距离，可以初始化为0（表示无法到达）。BFS可以一次找到1号点到所有点的最短距离。从1号开始找相邻节点，每次相邻点的距离+1，那么其他每一个点第一次被访问到的时候的距离就是最短的。（回忆一下最短路径的时候边的权值为1的时候的BFS解法）。同时这个数组可以取代visited数组，访问相邻节点的时候如果该节点的最短距离不是0，说明前面更早被访问过，有更短的路线，这次就不访问了。

也可以初始化最短路径数组为最大整数，程序中访问到的时候判断是否小于前面找到的最短，如果大于等于就不要往下走了。

# 改进

节点1到每一个点可能都有不止一条路到。前面的最短路的算法算的肯定是最短的一条。但是问题来了，我们希望的其实是对于每一个节点保存两个最短：1个是奇数步的最短，1个是偶数步的最短。

所以，修改BFS标准算法，但是每一次找到一个节点的时候，看看是偶数步还是奇数步，分开保存最短路径，并且分开设置visited数组，从而使得我们对每一个点可能访问两次，一个是奇数步最短，一个是偶数步最短。

如果实在想不清楚，也可以这么想：就相当于走2次BFS，一次统计到1号点最短的奇数步的最短路径；一次统计偶数步的最短路径。有环，所以节点1到任意一个点可能不止一条路，所以奇数步和偶数步都可能存在一个最短。

# 设计数据结构1

- 这里有 $10^5$ 个点，如果用静态二维数组的矩阵结构保存点和点之间的连接，空间是 $4 \times 10^5 \times 10^5$ ， $10^6$ 就是1M，肯定空间不够。但是实际上，所有的边的数目不超过 $10^5$ 。
- 邻接点列表： `vector <vector<int> > edges(INTMAX);`
- 这是动态二维数组， (INTMAX)是初始化外面的一维为100002。原因是二维如果都动态，操作比较麻烦。现在相当于开出了100002个初始化为空的数组（每一个数组保存所有当前节点的相邻点的编号），也就是开出100002行，但是每一行是空的。这样我们访问节点a的相邻节点组成的动态数组就可以用edges[a]来访问了， edges[a] 是一个vector数组，里面放着所有节点a的相邻节点。比如a b有连接就可以：

```
edges[a].push_back(b);
```

```
edges[b].push_back(a);
```

这时候，edges这个二维动态数组读入所有的边数据之后，总共大约就是 $2 \times 10^5$ 个整数。



看程序

# 小tips—双端队列

- 普通的queue，后面插入，前面取出，严格的先进先出
- 和queue类型非常像，不过是双向的，双向取出和双向添加，这样可以让优先级高的“插队”

```
#include <deque>
```

```
//a) 构造函数
```

```
deque<int> ideq
```

```
//b)增加函数
```

```
ideq.push_front(x):双端队列头部增加一个元素x
```

```
ideq.push_back(x):双端队列尾部增加一个元素x
```

```
//c)删除函数
```

```
ideq.pop_front():删除双端队列中最前一个元素
```

```
ideq.pop_back():删除双端队列中最后一个元素
```

```
ideq.clear():清空双端队列中元素
```

```
//d)判断函数
```

```
ideq.empty():向量是否为空，若true,则向量中无元素
```

```
//e)大小函数
```

```
ideq.size():返回向量中元素的个数
```

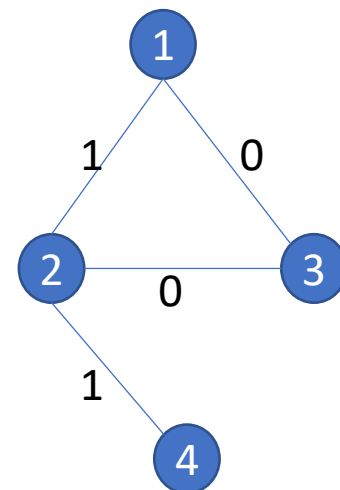




0/1BFS

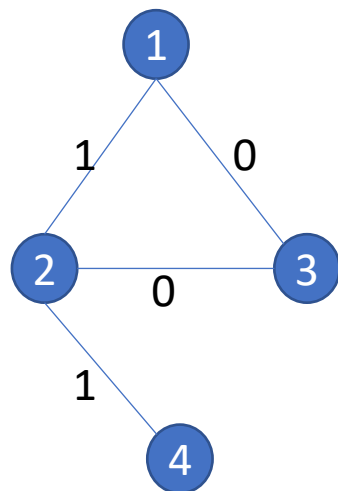
# 0/1BFS

- 如图：如果所有节点直接的路径长度不是1就是0，如何一次BFS找到1号点到所有点的最短路径？
- 首先需要有一个数组记录每一个点到节点1的最短路径。然后从节点1开始BFS。把每一个节点的最短距离更新到这个数组。
- 如果所有边的长度都是1，那么直接BFS找最短，第一层访问2，3，第二层是4。这样就可以更新最短路径数组，节点2，3的最短距离是1，4的是2。注意这里每一个节点只会被访问一次，因为BFS是按照层次的，先访问的肯定是最短路径。标准BFS找最短，最短路径长度其实就是经过的节点的个数，或者说相邻的层数。
- 现在？有的边为0，显然，原来的BFS层次不能直接使用，如果按照上述算法，那么节点4的最短是2，但是如右图，明显是1，从节点1到3到2再到4.长度为1。这个时候层数不等于最短路径的长度。



# 0/1BFS

- 标记每一个点的当前最短路径（初始化除了开始节点，其他的都最大），只有通过当前节点+0或者+1能够到达相邻节点比上次找到的相邻节点的最短路径还短，才会把把相邻节点入队列的头部或者尾部。
- 用双向队列deque来处理相邻节点：
- 每一次如果是相邻节点是长度为0的点，就加到队列前端，为1的放到队列最后。这样BFS每一次都是从队列前端找，优先处理的就是相邻节点长度为0的点。
- 或者可以理解长度为0的相邻点，其实和当前的点就是一个层次的了，当前点处理后马上就会处理这些点。先找到的就是最短。
- 如图：现在边长度分别是0，1，双端队列的访问次序就是：节点1，节点3，节点2，节点4，长度为1。所以如图dis数组所标记的，4个节点的最短路径得到了正确的答案。注意，处理3号节点的时候。3号节点的相邻节点有节点2和节点1，节点1的步数是0和dis[0]一样，不入队；节点2的步数是0，比dis[2]小，由于距离是0，插队到队列！！节点4也插队，最后队列里面的原来的步数为2的节点2，一直等到插队节点处理完才处理，这个时候显然比dis[2]要大。



1		dis[1]=0
3	2	dis[3]=0
2	2	dis[2]=0
4	2	dis[4]=1
	2	dis[2]不修改

# 算法

初始化d数组元素为最大

$d[s] = 0$ ; //s为出发点

`deque<int> q;`

`q.push_front(s);`

`while (!q.empty()) {`

`int v = q.front();`

`q.pop_front();`

    遍历v的所有相邻节点u{

`if (d[v] + w < d[u])` { //w为1或者0, 表示u和v的距离

            //只有通过v到u的距离比原来的d[u]小才入队列, 更新d[u]

`d[u] = d[v] + w;`

`if (w == 1)`

`q.push_back(u);`

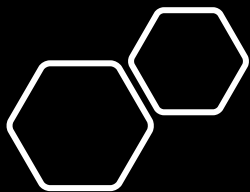
`else`

`q.push_front(u);` //插队

`}`

`}`

`}`



# USACO 2019 December Contest, Silver Milk Visits

- 先看题，理解题目
- $N$  个农场，用  $N-1$  条道路连接，构成一棵树（也就是说，所有农场之间都互相可以到达，并且没有环）
- 注意：无环意味着两点之间只有一条道路，或者说一个节点在一条道路上只会被访问一次，所以在DFS BFS的时候可以用visited数组去控制一个节点只能被访问一次。

分析：节点，道路，首先想到图，无环

5 5

HHGHG

1 2

2 3

2 4

1 5

**1 4 H**

**1 4 G**

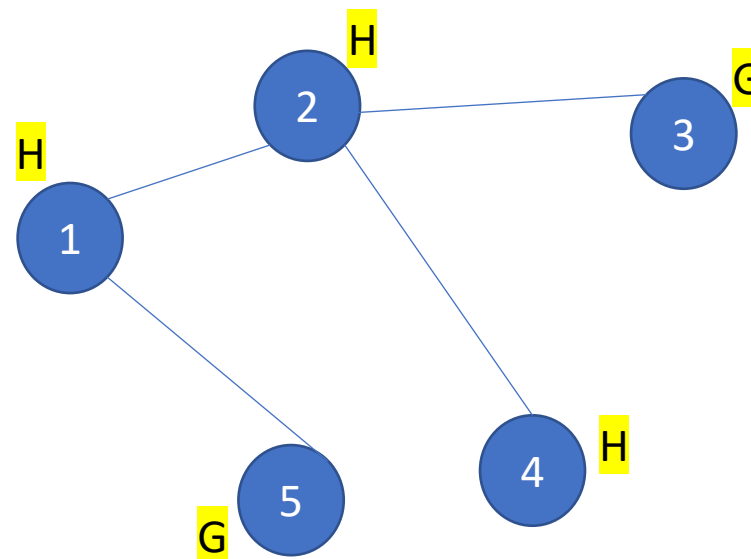
**1 3 G**

**1 3 H**

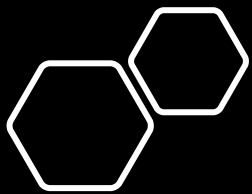
**5 5 H**

输出

**10110**



BFS或者DFS很容易找到两个节点之间经过的所有节点，比如1 4 H, 路径就是1 2 4，只要判断这几个节点是否有H就能够判断。由于无环BFS DFS很容易，节点不会多次被访问，只能访问一次（因为无环），从起始点出发，找相邻，只要到达终节点就退出，经过的就是路线上的点。



多次DFS太慢只  
能一次DFS把信  
息都找到，联通？

- 最简单的办法当然是DFS，对于每一个朋友都可以DFS一遍，找路径上的农场有没有匹配的奶牛。但是这样只能得一半的分，爆掉。
- 怎么办呢？还是前面的思路，肯定只能DFS常数级别的次数，比如一次。这个反向思考一下：我们DFS一遍，把能够只用一种牛奶到达的作为一个联通，那么下次检查两个点，如果这两个点和他要喝的奶不一样，我们就要看这两个点是不是在一个联通，如果在，说明他们之间不会出现我们想要的奶，返回0。
- 编程就简单了，和前面的寻找联通component的算法差不多，只不过遇到flag（奶牛品种）不一样的就退出。其实就是找同样的奶牛品种的联通图。
- 这里用DFS，其实BFS，并查集都可以

# 如图3个component (单元)

5 5

HHGHG

1 2

2 3

2 4

1 5

1 4 H 节点1 4出现了H, ok

1 4 G 节点1 4没有G, 1和4是一个单元, 说明中间没有G的点, not ok

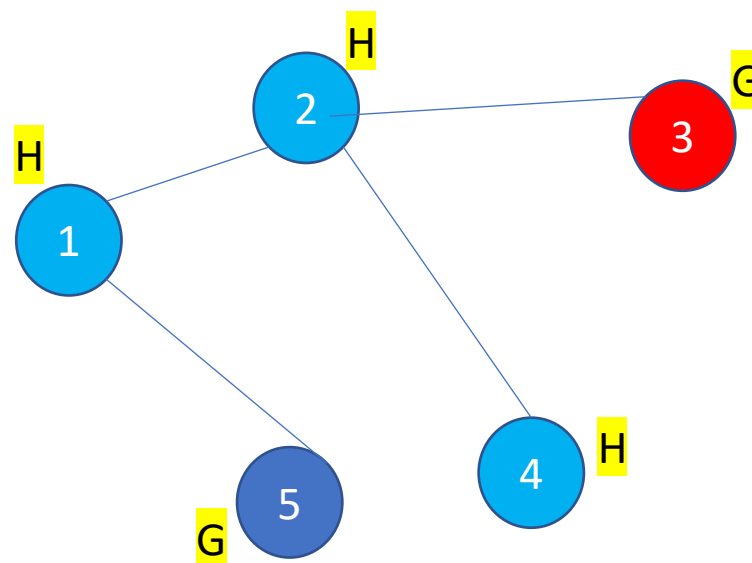
1 3 G 节点1 3出现了G, ok

1 3 H 节点1 3出现了H, ok

5 5 H 节点5不是H, 而且5 5 在一个单元, 说明中间没有H的点, not ok

输出

10110







看看程序

```
void getcomponent(int index,char milkflag,int cindex)
//find all points connected with index and has same milkflag, set component to cindex--componentindex
{
    for (int i=0;i<edges[index].size();i++)
    {
        int next=edges[index][i];
        if (!visited[next]) // 这个visited数组应该也可以用component数组替代
        {
            visited[next]=true;
            if (flag[next]==milkflag)
            //和标准联通最大的区别在这里，如果下一个相邻节点的标记不是我们这次要找的，那么就不是合法的相邻节点
            {
                component[next]=cindex;
                getcomponent(next,milkflag,cindex);
            }
        }
    }
    return ;
}
```

tips

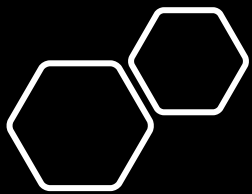
任意正整数和0异或的结果是自己。



比如如果答案是很多整数异或的结果，比如结果存放在`int ans;`里面，`ans`初始化应该为0，这样对于每次的结果都和`ans`去做异或 $\wedge$ 操作。

# 下节课预习

- NOIP2009细胞分裂 <https://www.luogu.com.cn/problem/P1069>
- NOIP2006 Jam的计数法 <https://www.luogu.com.cn/problem/P1061>
- NOIP2004火星文 <https://www.luogu.com.cn/problem/P1088>



# 作业1

- 迷宫 <https://www.luogu.com.cn/problem/P1141>

# 作业2

- 课堂讲的两个例题你可以自己从头写出来么？

由易到难，思维体系训练  
实战结合，创新协作培养  
兴趣导向，未来职业引领

<https://www.35tang.com>



扫码关注公众号

<https://www.三五堂.com>



添加辅导老师