[9185] 假期计划(holiday)

小熊的地图上有 n 个点，其中编号为 1 的是它的家、编号为 2,3,...,n 的都是景点。部分点对之间有双向直达的公交线路。如果点 x 与 z1、z1与 z2、……、zk−1 与 zk、zk 与 y 之间均有直达的线路，那么我们称 x 与 y 之间的行程可转车 k 次通达；特别地，如果点 x 与 y 之间有直达的线路，则称可转车 0 次通达。

很快就要放假了，小熊计划从家出发去 4 个不同的景点游玩，完成 5 段行程后回家：家 → 景点 A → 景点 B → 景点 C → 景点 D → 家且每段行程最多转车 k 次。转车时经过的点没有任何限制，既可以是家、也可以是景点，还可以重复经过相同的点。例如，在景点 A → 景点 B 的这段行程中，转车时经过的点可以是家、也可以是景点 C，还可以是景点 D → 家这段行程转车时经过的点。

[9185] 假期计划(holiday)

假设每个景点都有一个分数，请帮小熊规划一个行程，使得**小熊访问的四个不同景点的分数之和最大**。

输入第一行包含 3 个正整数 n,m, k，分别表示地图上点的个数、双向直达的点对数量、每段行程最多的转车次数。

第二行包含 n−1个正整数，分别表示编号为 2,3,...,n 的景点的分数。接下来 m 行，每行包含两个正整数 x, y，表示点 x 和 y 之间有道路直接相连，保证 1≤x,y≤n，且没有重边，自环。

输出一个正整数，表示小熊经过的 4 个不同景点的分数之和的最大值。

对于所有数据，保证 5≤n≤2500,1≤m≤10000,0≤k≤100，所有景点的分数 1≤si≤1018。保证至少存在一组符合要求的行程。

[9185] 假期计划(holiday)

输入样例1：

8 8 1

9 7 1 8 2 3 6

1 2

2 3

3 4

4 5

5 6

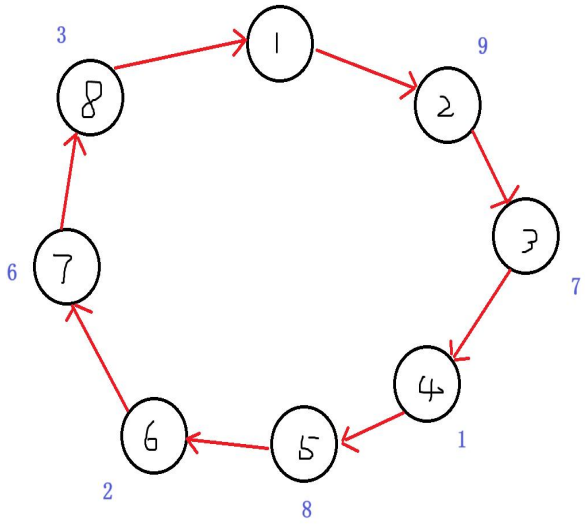
6 7

7 8

8 1

输出样例1：

27



样例1解释 当计划的行程为 1→2→3→5→7→1时，4 个景点的分数之和为 9+7+8+3=27，可以证明其为最大值。

行程 1→3→5→7→8→1的景点分数之和为 24、行程 1→3→2→8→7→1的景点分数之和为 25。

它们都符合要求，但分数之和不是最大的。

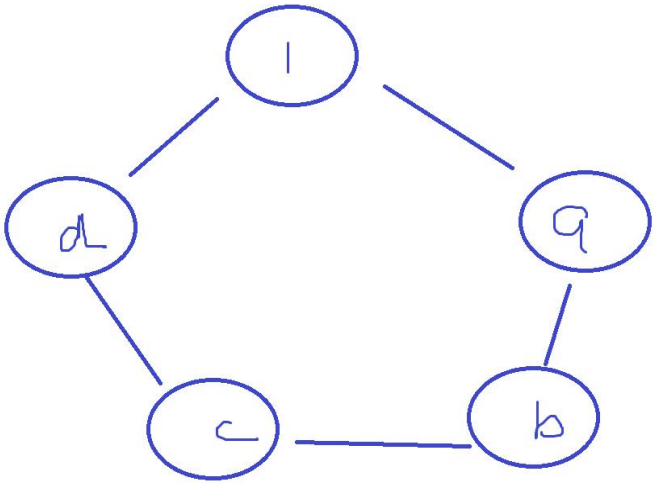
行程 1→2→3→5→8→1 的景点分数之和为 30，但其中 5→8至少需要转车 2 次，因此不符合最多转车 k=1次的要求。

行程 1→2→3→2→3→1的景点分数之和为 32，但游玩的并非 4 个不同的景点，因此也不符合要求。

[9185] 假期计划(holiday)

题目中最重要的信息是“k次转乘”，也就是和点 u的距离小于等于k+1的点都可以直接到达。所以可以用 bfs进行暴力连边，于是原问题转化为：求一个长度为 5且 包括节点 1的环，使得环上的 5个点点权之和最大。

但是其实所谓的“暴力连边”还可以优化。因为环上只有 5 个点，所以其实有很多连边是不必要的。可以用一个vector来存储“遍历到目前有用的边”，接下来再暴力枚举一下四个点。枚举 B和 C，再判断如果 B 和 C 有连边，A 和 D 和 1有连边，四个点不相同。其中 A 和 D 是与 原来两个点有连边的点。



时间复杂度为 $O(n^4)$

如何解决？

空间换时间！预处理！

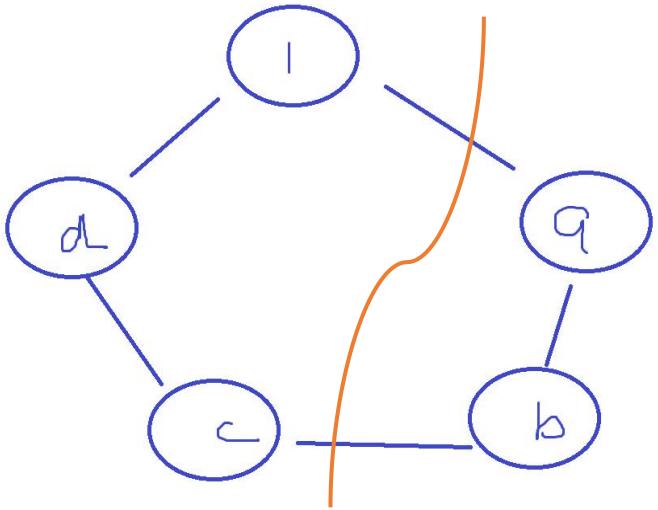
[9185] 假期计划(holiday)

我们定义 $f(u)$ 表示 u **可达**，且**在家附近**的**权值最大**的景点。

第一步：我们预处理出 $f(u)$。

第二步：直接 n^2 枚举，循环确定景点 b 和 c （$b \neq c$），然后记 $a = f(b)$，$d = f(c)$，然后试图用 $w(a) + w(b) + w(c) + w(d)$ 更新答案，其中 $w(u)$ 表示景点 u 的权值。

发现重复性的细节是存在问题的，比如：会出现 $f(b) = c$ 的情况，此时让 $a = f(b)$ 会导致 $a = c$，这是不允许的。



[9185] 假期计划(holiday)

不过，贪心思想还是不变的：a 应该尝试更换为 u 可达，且在家附近的**权值第二大**的景点。

更换定义，$f(u,k)$ 表示 u 可达，且在家附近的权值第 k 大的景点。

当发现 $f(b,1)=c$ 时，将 a 设置为 $f(b,2)$ 即可。

当发现 $f(c,1)=b$ 时，将 d 设置为 $f(c,2)$ 即可。

发现并没有完全解决：如果这样处理后的 a 和 d 仍然重复怎么办?

还是贪心思想，考虑把 a 或者 d 换成更小的那个比较一下就行，具体来说，比如原先 $a=f(b,2)$，就把 a 下调为 $f(b,3)$；或者原先 $d=f(c,2)$，就把 d 下调为 $f(c,3)$，然后比较两种方案哪种更好就行了。

重复性解决了，但是还有个细节：如果原先 $a=f(b,1)$，发现 $a=d$，我们想下调 a。是直接把 a 设置成 $f(b,2)$ 吗? 错误，我们还需要检查一下 $f(b,2)$ 是否等于 c，如果等于 c 还需要继续下调到 $f(b,3)$。下调 d 同理。

至于原先 $a = f(b, 2)$ 为啥下调 a 不需检查？因为如果原先 $a = f(b, 2)$ 了说明 $f(b, 1)$ 已经等于 c 了。所以 $f(b, 3)$ 肯定什么问题都没有。

如果直接这么写是没有问题的，但是麻烦了。下面是一种更好写的处理方法：

直接分别枚举 a 分别作为 $f(b, 1)$，$f(b, 2)$，$f(b, 3)$ 和 d 分别作为 $f(c, 1)$，$f(c, 2)$，$f(c, 3)$ 组成的共九种情况，检查互异性然后更新答案即可。

需要提前预处理出 $f(u, k \leq 3)$，其实只要在最开始 bfs 预处理的同时维护一下就行了。

注意某些点**可达，还在家附近**的点数可能没有 3 个，因此注意类似访问 $f(b, 3)$ 时产生的越界问题。

如果枚举到某个 b，c，发现 b 或者 c 有对应点数不超过 3 的情况时，这组 b，c 不一定可以生成一组合法的解，属于正常现象。

题目保证全局上是有解的。

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 2510;
vector<int> edge[N], f[N];// f[u] 存放：可达 u 且可达 1 的前三大 v
int n, m, k, dist[N];
long long D[N], ans = 0;
bool w[N][N];// u, v 是否可达

queue<int> q;

bool cmp(int a, int b) {
    return D[a] > D[b];
}

void bfs(int x) {
    for (int i = 1; i <= n; i++) dist[i] = -0x3f3f3f3f;
    while (!q.empty()) q.pop();
    dist[x] = 0;
    q.push(x);
    while (!q.empty()) {
        int u = q.front(); q.pop();
        if (x != u) {
            w[x][u] = true;
            if (x != 1 && w[1][u]) {//1能到达u
                f[x].push_back(u);
                sort(f[x].begin(), f[x].end(), cmp);
                // 这里 sort 元素数量不超过 3，效率可看做常数
                if (f[x].size() > 3) f[x].pop_back();
            }
        }
```

```
31          if (dist[u] > k) continue;
32          for (auto i : edge[u])
33              if (dist[i] == -0x3f3f3f3f)
34                  dist[i] = dist[u] + 1, q.push(i);
35      }
36  }
37
38  int main() {
39      scanf("%d%d%d", &n, &m, &k);
40      for (int i = 2; i <= n; i++) scanf("%lld", &D[i]);
41      while (m--) {
42          int u, v;
43          scanf("%d%d", &u, &v);
44          edge[u].push_back(v);
45          edge[v].push_back(u);
46      }//建图
47      for (int i = 1; i <= n; i++) bfs(i);//bfs预处理
48
49      for (int b = 2; b <= n; b++)
50      for (int c = 2; c <= n; c++)
51      if (w[b][c]) {//b、c间可到达
52          //// 其他不等关系天然满足了，只有这三组需要检验：
53          // 天然满足的不等关系：a != b，b != c，c != d，
54          for (auto a : f[b]) //f[b][c]
55          for (auto d : f[c])
56          if (a != b && a != c && a != d && b != c && b != d && c != d)
57              ans = max(ans, D[a] + D[b] + D[c] + D[d]);
58
59      }
60      printf("%lld\n", ans);
61      return 0;
62  }
```

[9185] 假期计划(holiday)

轩轩是 1 号工人。现在给出 q 张工单，第 i 张工单表示编号为 a_i 的工人想生产一个第 L_i 阶段的零件。轩轩想知道对于每张工单，他是否需要给别人提供原材料。他知道聪明的你一定可以帮他计算出来！

[9185] 假期计划(holiday)

轩轩是 1 号工人。现在给出 q 张工单，第 i 张工单表示编号为 a_i 的工人想生产一个第 L_i 阶段的零件。轩轩想知道对于每张工单，他是否需要给别人提供原材料。他知道聪明的你一定可以帮他计算出来！

[9185] 假期计划(holiday)

轩轩是 1 号工人。现在给出 q 张工单，第 i 张工单表示编号为 a_i 的工人想生产一个第 L_i 阶段的零件。轩轩想知道对于每张工单，他是否需要给别人提供原材料。他知道聪明的你一定可以帮他计算出来！

[7330] 加工零件

轩轩是 1 号工人。现在给出 q 张工单，第 i 张工单表示编号为 a_i 的工人想生产一个第 L_i 阶段的零件。轩轩想知道对于每张工单，他是否需要给别人提供原材料。他知道聪明的你一定可以帮他计算出来！

输入

第一行三个正整数 n，m 和 q，分别表示工人的数目、传送带的数目和工单的数目。

接下来 m 行，每行两个正整数 u 和 v，表示编号为 u 和 v 的工人之间存在一条零件传输带。保证 u≠v。

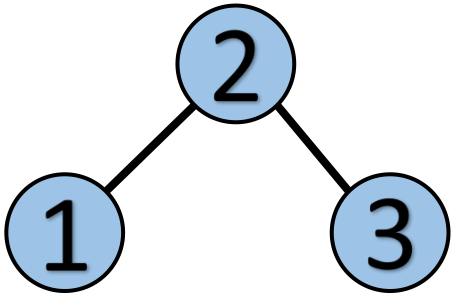
接下来 q 行，每行两个正整数 a 和 L，表示编号为 a 的工人想生产一个第 L 阶段的零件。

输出

共q行，每行一个字符串“Yes”或者“No”。如果按照第 i 张工单生产，需要编号为 1 的轩轩提供原材料，则在第 i 行输出“Yes”否则输出“No”。注意不含引号。

$1 \le n,m,q \le 10^5$，$1 \le L \le 10^9$

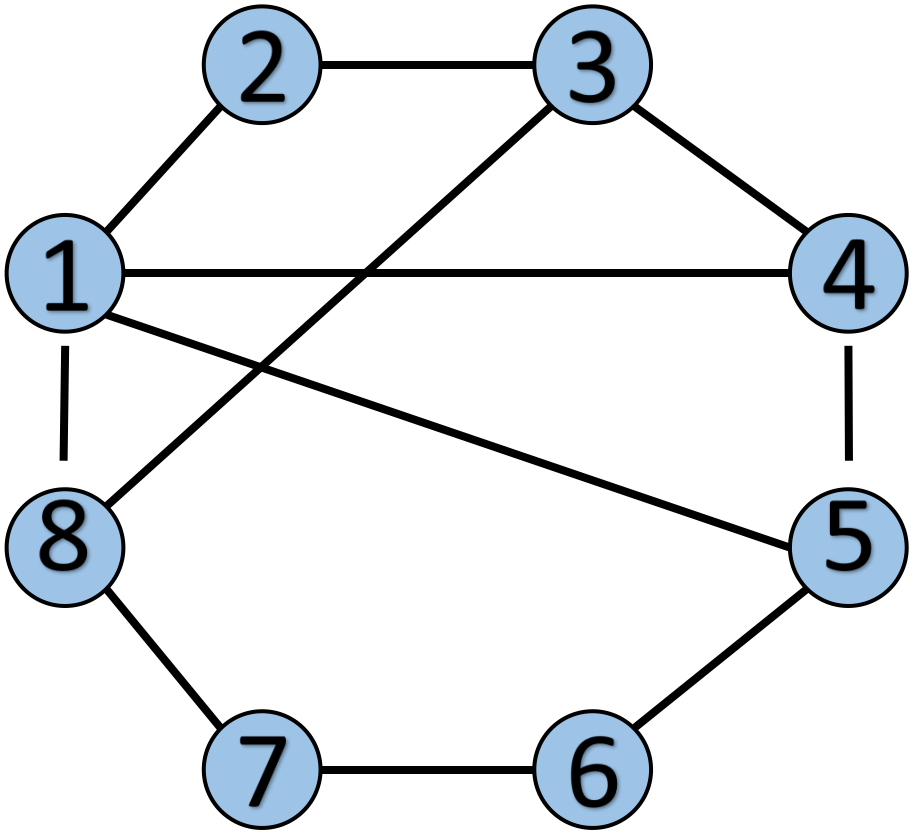
分析样例



编号为 a 的工人想生产一个第 L 阶段的零件。

a	L	out
1	1	No
2	1	Yes
3	1	No
1	2	Yes
2	2	No
3	2	Yes

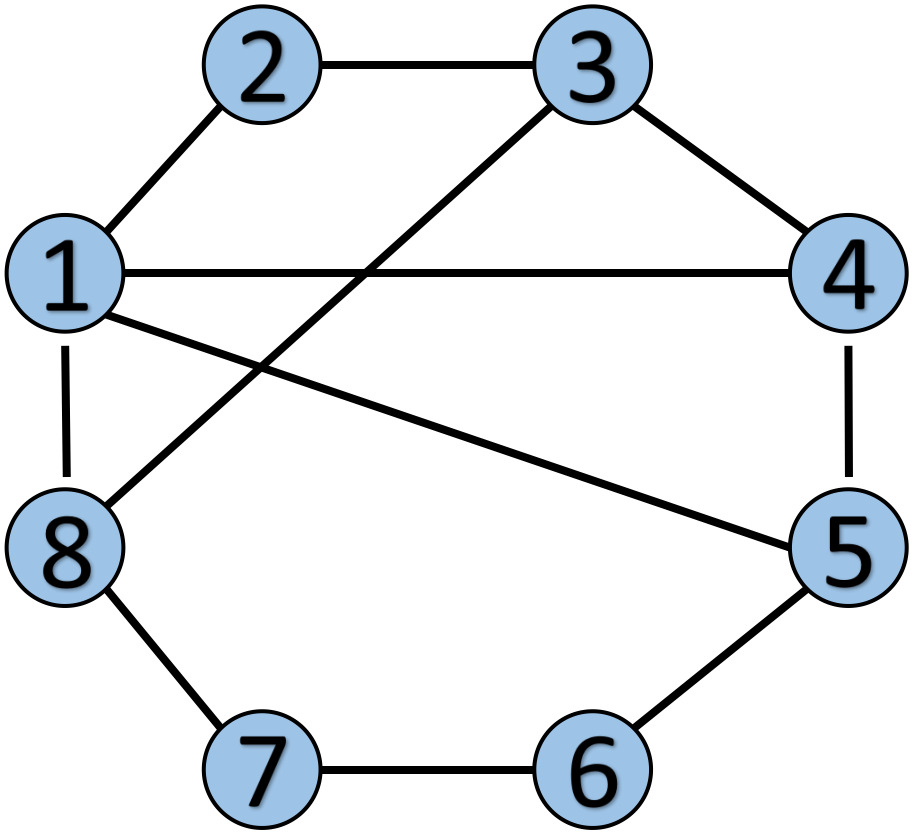
举例理解

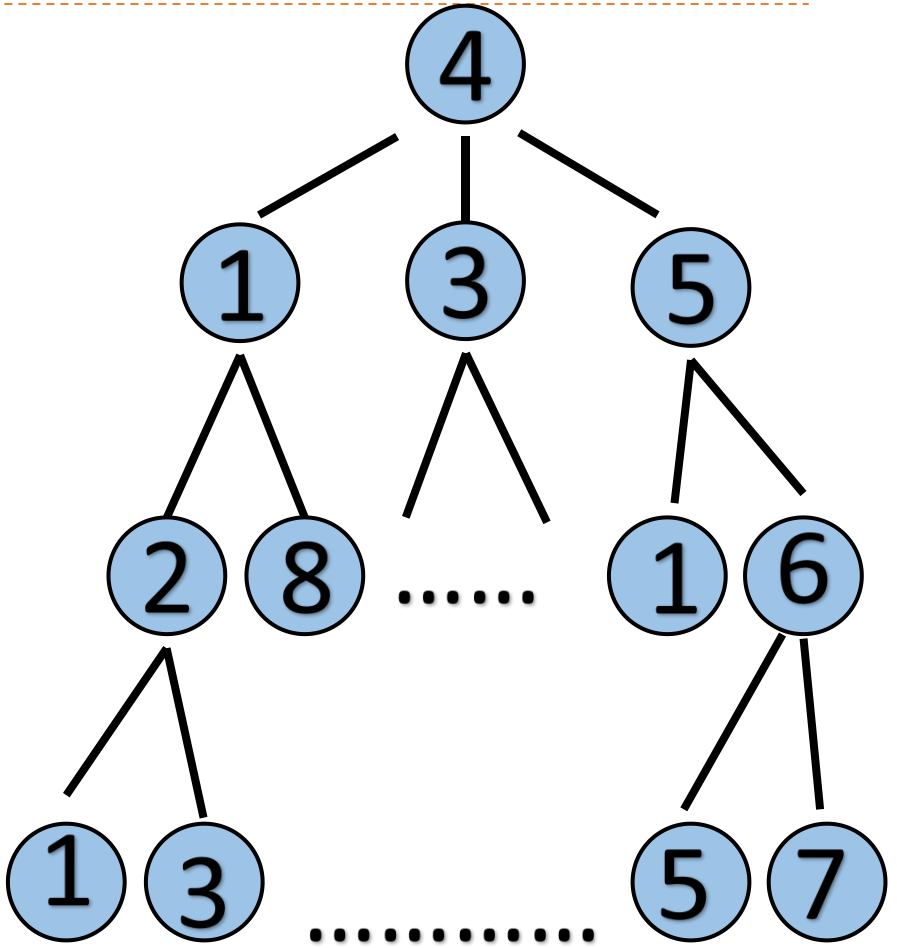


a=4　　L=2

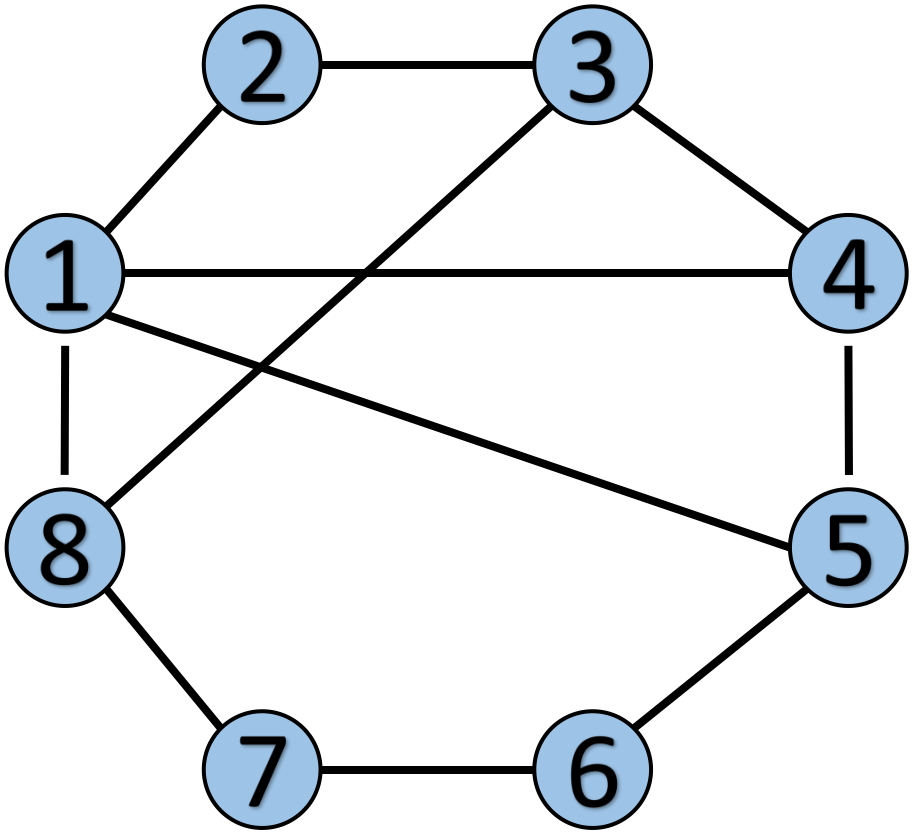
Out：Yes

需要关心全局么？





问题退化为



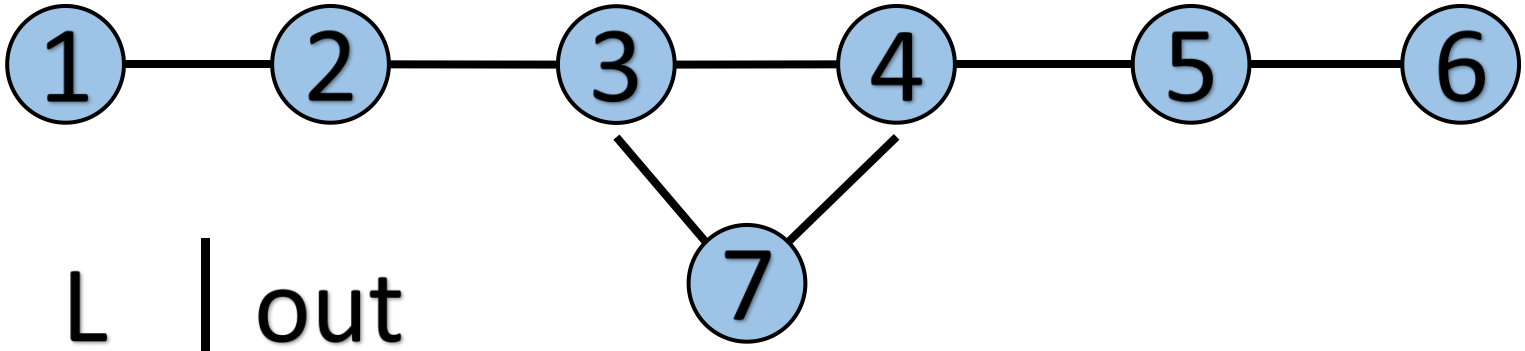
- 从 a点开始；
- 恰好行走 L步；
- 是否能恰好走到 1点。

又等价于____？

等价嘛？

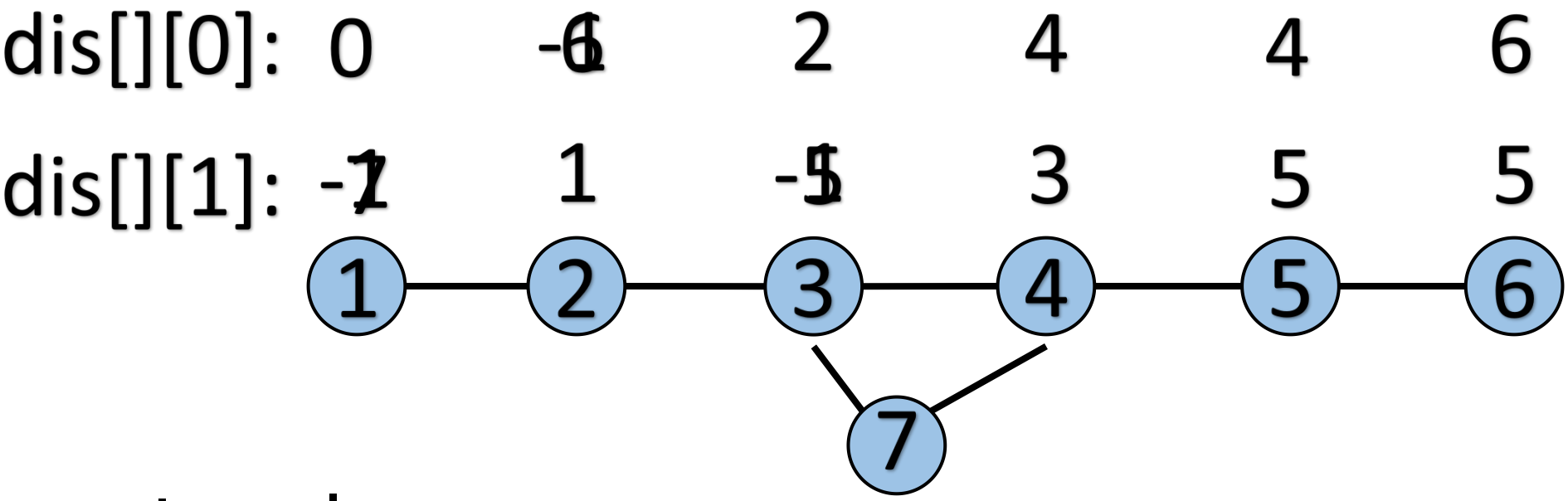
问题解决了么?

dis[]: 0 1 2 3 4 5



a	L	out
5	3	No
5	8	Yes
5	9	?

兵来将挡，水来土掩



a	L	out
5	9	Yes

代码——BFS部分

由于我们可以在某一条边上玩折返跑，因此只要有从 1 号点出发到 a 距离小于等于 L 且奇偶性与 L 相等的路径
那么就有恰好经过 L 条边抵达 a 号点的方案

定义 d[j][0] 表示经过偶数条边
d[j][1]为经过奇数条边抵达 j 号的最短距离

用 bfs 跑一遍最短路即可，每个点最多入队两次

```cpp
int main(){
    int n,m,q;
    cin>>n>>m>>q;
    for(int i=1;i<=m;i++){
        int a,b;
        scanf("%d %d",&a,&b);
        ve[a].push_back(b);//建图 邻接表加边
        ve[b].push_back(a);
    }
    bfs();
    for(int i=1;i<=q;i++){
        int a,b;
        scanf("%d %d",&a,&b);
        if(b%2==1){
            if(d[a][1]<=b)printf("Yes\n");
            else printf("No\n");
        }
        else if(b%2==0){
            if(d[a][0]<=b)printf("Yes\n");
            else printf("No\n");
        }
    }

    return 0;
}
```

```cpp
#include<bits/stdc++.h>
using namespace std;

const int N=1e5;
const int INF=0x3f3f3f3f;
int d[N][2];
vector<int> ve[N];// 邻接表
```

```
void bfs(){// 预处理（BFS）
    queue<int>q;
    q.push(1);
    memset(d,INF,sizeof d);// 初距离始化为正无穷
    d[1][0]=0;// 初始化队列起点
    while(!q.empty()){
        int u=q.front();
        q.pop();
        for(int i=0;i<ve[u].size();i++){//遍历邻接表表邻边
            int v=ve[u][i];// 取出邻边编号
            if(d[v][0]>d[u][1]+1||d[v][1]>d[u][0]+1){
                if(d[v][0]>d[u][1]+1)// // 如果走一步的长度大于dist[j][type]+
                    d[v][0]=d[u][1]+1;
                if(d[v][1]>d[u][0]+1)/// 注：由于走一步就会变一下奇偶性，
                    d[v][1]=d[u][0]+1;
                q.push(v);//  刷新并将点更新到队列中
            }
        }
    }
}
```

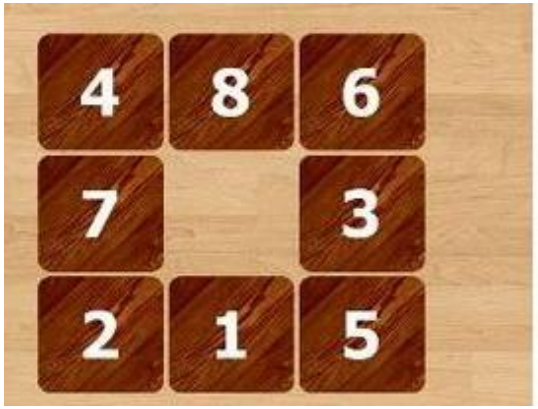
[7431]八数码

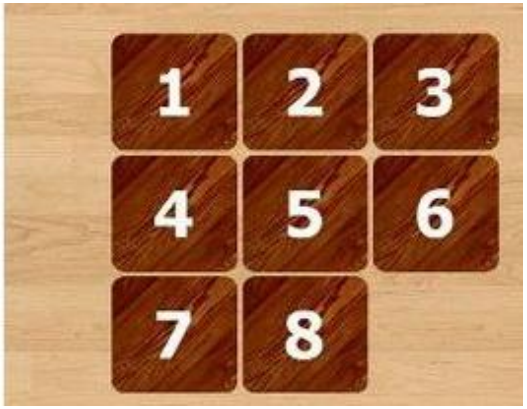
例如，如果初始网格如下所示：

4 8 6

7 x 3

2 1 5





则输入为：1 2 3 x 4 6 7 5 8

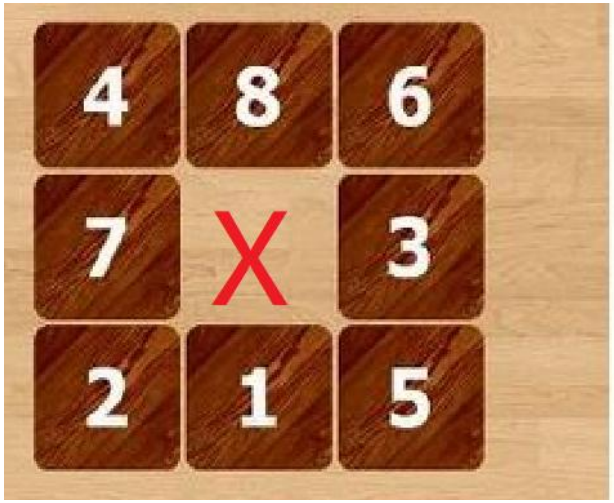
输出占一行，包含一个整数，表示最少交换次数。

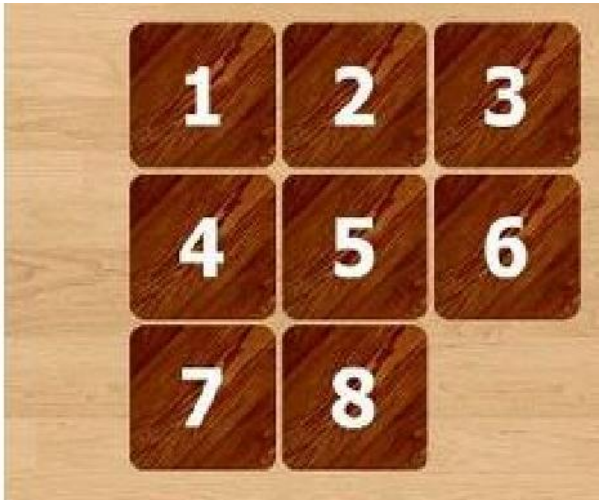
如果不存在解决方案，则输出 -1-1。

样例输入　　4 8 6 7 x 3 2 1 5

样例输出　　22

分析:基本思路





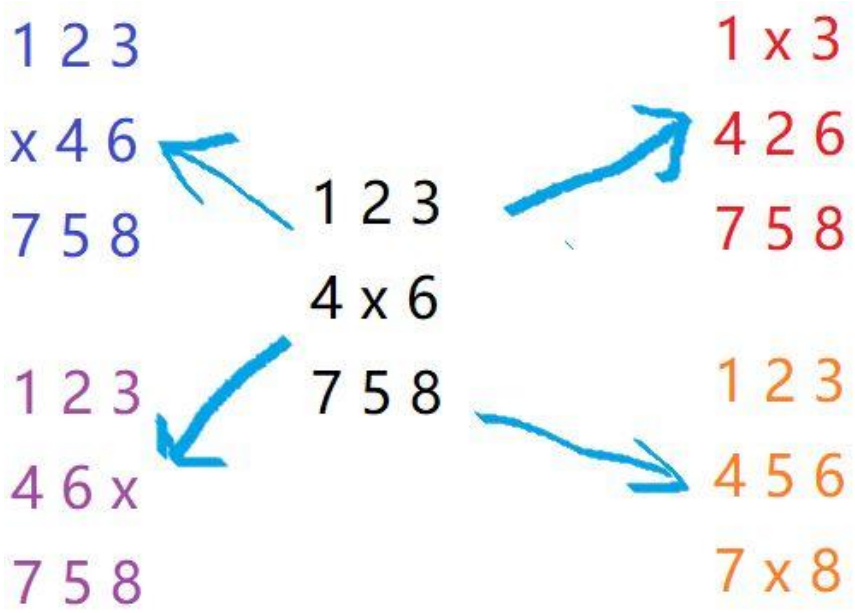
用BFS

每次将x与上下左右的一个交换位置，求最小交换次数，若无法转换为目标状态，则输出-1

如果把每一种状态当作1个节点，每一移动一个位置，就是花费一个代价从一个状态转移到下一个状态。目标是从起点到终点需要花费最小的代价是多少。

从初始状况移动到目标情况 —> 求最短路

分析:移动方向



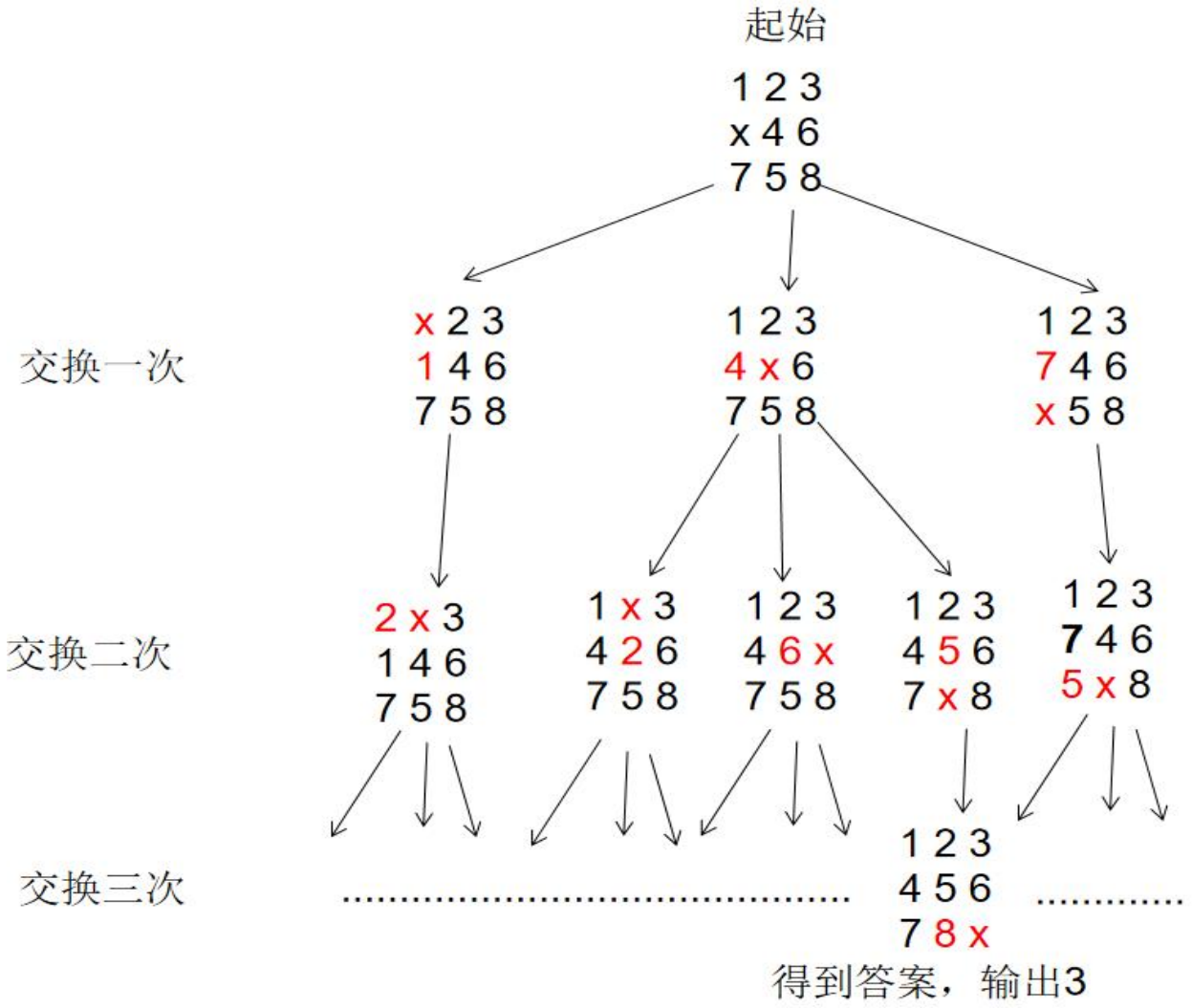
移动方式

```
int dx[4] = {1, -1, 0, 0},
int dy[4] = {0, 0, 1, -1};
```

转以后:

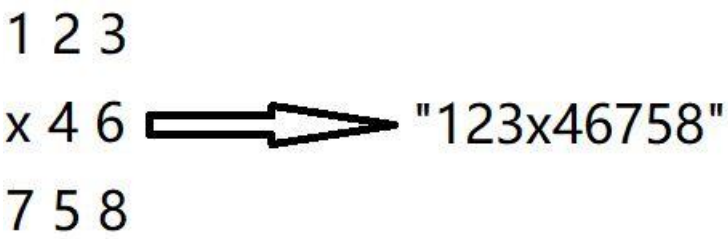
```
a = x + dx[i],b = y + dy[i]
```

分析:基本思路


起始
1 2 3
x 4 6
7 5 8
交换一次
交换二次
交换三次
x 2 3
1 4 6
7 5 8
1 2 3
4 x 6
7 5 8
1 2 3
7 4 6
x 5 8
2 x 3
1 4 6
7 5 8
1 x 3
4 2 6
7 5 8
1 2 3
4 6 x
7 5 8
1 2 3
4 5 6
7 x 8
1 2 3
7 4 6
5 x 8
1 2 3
4 5 6
7 8 x
得到答案，输出3

分析:状态表示

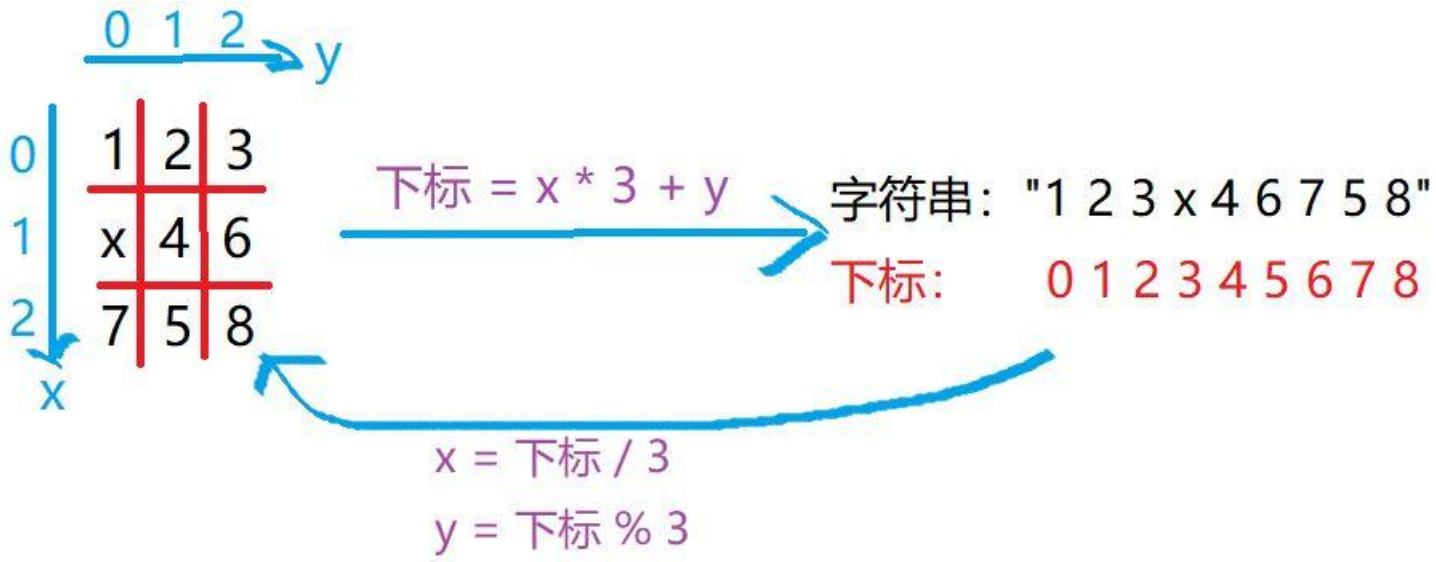
将 “3*3矩阵” 转化为 “字符串”




```
string end = "12345678x";//定义目标状态
queue<string> q;//定义队列和dist数组
map<string, int> d;//记录每个状态的距离
```

分析:状态表示

5、矩阵与字符串的转换方式




```
int k = t.find('x');
int x = k / 3, y = k % 3;
swap(t[k], t[a * 3 + b]);
```

交换字符串t中的字符（x与四领域的字母）

算法思路

用一个队列保存当前获得的序列

用一个哈希表保存各个序列与对应的交换次数。

从队列中取出队头这个序列，计算出这个序列通过交换能得到的序列。如果能到得的序列是个新序列（哈希表中没有这个序列），就把这个新序列放入队尾，哈希表中记录新序列对应的交换次数。

如果在上述过程中得到了结果序列，则输出交换次数，结束。

如果最终没有得到结果序列。输出-1。

```
1   #include <iostream>
2   #include <algorithm>
3   #include <queue>
4   #include <map>
5   using namespace std;
6
7   int bfs(string start)
8   {
54
55  int main()
56  {
57      string c, start;
58      //输入起始状态
59      for(int i = 0; i < 9; i++)
60      {
61          cin >> c;
62          start += c;
63      }
64      cout << bfs(start) << endl;
65      return 0;
66  }
```

```
 7  int bfs(string start)
 8  {
 9      //定义目标状态
10      string end = "12345678x";
11      //定义队列和dist数组
12      queue<string> q;
13      map<string, int> d;
14      //初始化队列和dist数组
15      q.push(start);
16      d[start] = 0;
17      //转移方式
18      int dx[4] = {1, -1, 0, 0}, dy[4] = {0, 0, 1, -1};
19
20      while(q.size())
21      {
51      //无法转换到目标状态，返回-1
52      return -1;
53  }
```

```
20      while(q.size())
21      {
22          string t = q.front();
23          q.pop();
24          //记录当前状态的距离，如果是最终状态则返回距离
25          int distance = d[t];
26          if(t == end) return distance;
27          //查询x在字符串中的下标，然后转换为在矩阵中的坐标
28          int k = t.find('x');
29          int x = k / 3, y = k % 3;
30
31          for(int i = 0; i < 4; i++)
32          {
50      }
51      //无法转换到目标状态，返回-1
52      return -1;
53  }
```

```
31          for(int i = 0; i < 4; i++)
32          {
33              //求转移后x的坐标
34              int a = x + dx[i], b = y + dy[i];
35              //当前坐标没有越界
36              if(a >= 0 && a < 3 && b >= 0 && b < 3)
37              {
38                  //转移x
39                  swap(t[k], t[a * 3 + b]);
40                  //如果当前状态是第一次遍历，记录距离，入队
41                  if(!d.count(t))
42                  {
43                      d[t] = distance + 1;
44                      q.push(t);
45                  }
46                  //还原状态，为下一种转换情况做准备
47                  swap(t[k], t[a * 3 + b]);
48              }
49          }
50      }
51      //无法转换到目标状态，返回-1
```

今天的课程结束啦……

下课了…
同学们再见！