本次实验的解决方式：

显然本题中，只有对应位置全部相同的两行才能在k次翻转之后有机会同时变为全1（导通）状态。因此先遍历一遍整个矩阵，用哈希表把所有出现的行的排列情况存在一个类数组里，该类包含两个成员变量，一个是行的1和0的排列顺序，另一个是该顺序在矩阵中出现的总次数。

接下来，遍历这个数组，对于每一个元素，检查这一行中0的个数是不是小于等于翻转次数k，如果不是，说明不可能经过k次翻转变为全1，直接遍历下一个元素；如果是，再查看k-（0的个数）是不是偶数，如果是，就说明可以把多余的反转次数全部消耗在翻转同一列上，因为显然，翻转偶数次之后同一列的值不发生变化，那么这一行就可以经过k次翻转变为全1。对数组中所有的成员都进行这样的遍历，找到能被反转为全1并且相同行数最多的一个即为最大导通数。

---优化-在前一个版本中，最后一次需要遍历全部的记录行排列情况的数组，因此做出优化，在读入行排列情况的时候，根据0的出现次数将其排入奇数组和偶数组中，这样在最后一次遍历的时候，对于k为偶数的情况，只需要遍历偶数组，对于k为奇数的情况，只需要遍历奇数组。这样不但减少了遍历次数，还减少了遍历过程中对k除去该行中0的个数之后剩余的反转次数是否是偶数的判断。

```cpp
#include <iostream>
#include <cstring> // 使用 memset 初始化
using namespace std;

int main() {
    int N, M, K;
    cin >> N >> M >> K;

    // 动态分配内存：仅考虑最多 N 行可能的模式
    int* patterns = new int[N];        // 存储每行的模式
    int* pattern_count = new int[N]; // 存储每种模式的出现次数
    int* zeros_count = new int[N];    // 存储每种模式中 0 的数量
    memset(pattern_count, 0, N * sizeof(int));

    int pattern_index = 0; // 已记录的模式数量

    // 输入矩阵并记录模式
    for (int i = 0; i < N; ++i) {
        int pattern = 0;
        for (int j = 0; j < M; ++j) {
            char c;
            cin >> c;
            if (c == '1') {
```

```
24                  pattern |= (1 << j); // 将对应列标记为 1
25              }
26          }
27
28          // 检查该模式是否已经存在
29          bool found = false;
30          for (int p = 0; p < pattern_index; ++p) {
31              if (patterns[p] == pattern) {
32                  pattern_count[p]++; // 模式出现次数增加
33                  found = true;
34                  break;
35              }
36          }
37          if (!found) {
38              // 新模式
39              patterns[pattern_index] = pattern;
40              pattern_count[pattern_index] = 1;
41
42              // 计算该模式中 0 的数量
43              int zeros = 0;
44              for (int j = 0; j < M; ++j) {
45                  if ((pattern & (1 << j)) == 0) {
46                      zeros++;
47                  }
48              }
49              zeros_count[pattern_index] = zeros;
50              pattern_index++;
51          }
52      }
53
54      // 遍历模式以找到最大行数
55      int max_rows = 0;
56      for (int p = 0; p < pattern_index; ++p) {
57          int zeros = zeros_count[p];
58          if (zeros <= K && (K - zeros) % 2 == 0) { // 满足条件
59              max_rows = max_rows > pattern_count[p] ? max_rows :
pattern_count[p];
60          }
61      }
62
63      // 输出结果
64      cout << max_rows << endl;
65
66      // 释放动态分配的内存
67      delete[] patterns;
68      delete[] pattern_count;
69      delete[] zeros_count;
```

```
70
71      return 0;
72  }
73
```