- 首先，认定0序列与1序列是按原来的顺序接受的。因为：设$i < j$假如存在着某一种延迟，使得原本

第i个1到了第j个1后面，则必然存在着一种延迟，使得i, j位置互换。因此，认定这个规则，这会对后面的分析有很大好处。

- 把序列中的0,1编号，O_i表示序列中第$i+1$个1的位置，Z_i表示序列中第$i+1$个0的位置。令$f_{i,j}$表示对于$Z_0 - Z_{i-1}, O_0 - O_{j-1}$中的元素以接收到，可能情况的个数。
- 设D为最长延迟，假设原序列K中已接受到的最右边的下标为p，可以证明它没有延迟为最优的情况。不妨设它有延迟d，则时间为$p+d$，设下一个接受的元素在K中的下标为q，如果$q > p$，如果$p + d < q + D$，若$q < p$，同样的，希望$p + d < p + D$，则最好的情况就是$d = 0$，得证。
- 如何判断$f_{i,j}$是否可以转换为$f_{i+1,j}$或$f_{i,j+1}$？假设插入0，则有$Z_i + d >= O_{j-1}$即可。即写为：

```
bool crz(int i, int j) {
  return i < zcnt && z[i] + d >= o[j - 1];

}
bool cro(int i, int j) {
  return j < ocnt && o[j] + d >= z[i - 1];
}
```

这样可以算出可能的情况数，但是，在计算$maxv, minv$是出现了问题，因为这种判断只顾眼前，只能说明这一步是可操作的，而并不知道下一步是否可以继续操作，使得$maxv, minv$的计算可能进入死循环或则得不到正确答案。

而写成这样：

```
bool crz(int i, int j) {
  return i < zcnt && z[i] + d >= o[j - 1] && (j == ocnt || o[j] + d >=
z[i]);

}
bool cro(int i, int j) {
  return j < ocnt && o[j] + d >= z[i - 1] && (i == zcnt || z[i] + d >=
o[j]);
}
```

在保证了这一步可以操作的同时又保证了至少下一步可以操作另外一个数。比如现在这一步插入1，如果1可以插入，并且0没有插完，如果$Z[i] + d < O[j]$,那么下一步就不可以插入0了，如果继续插入1，$Z[i] + d < O[j] < O[j+1]$，以后这个0就不可能插入了，于是，这一步不能插入1.

下面证明这种判断方式计算$maxv, minv$是不会陷入死循环。假设当前i, j，并且$cra(i,j), cro(i,j)$都是错误的。则有$O[j] + d < Z[i], Z[i] + d < O[i]$，这是矛盾的，则必然可以到达下一个状态，得证。

- 化简。第一个状态时$i = 0, j = 0$，到达下一个状态时，$Z[i] + d >= O[j-1]$判断是一定成立的，并且$O[j] + d >= Z[i])$保证了下一个状态的$Z[i] + d >= O[j-1]$成立，则可以化简为：

```
bool crz(int i, int j) {
    return i < zcnt && (j == ocnt || O[j] + d >= Z[i]);
}
bool cro(int i, int j) {
    return j < ocnt && (i == zcnt || Z[i] + d >= O[j]);
}
```

- 有了这些，就可以方便的编写程序了。