# 输入处理

问题输入的格式为：



其中，分别表示：

* 车厢数，
* 检查次数，
* 第次检查的信息，
* 第次检查车厢的信息

从题目中可知：

* 通行证只有个
* 通行证不能放在同一个车厢，所以在解中每个车厢都必须有一个通行证，而各个车厢通行证的状态（上层/下层）影响了解的有效性
* 检查员只抽查不同车厢的一层，所以对于，它的非零元素个数总是不大于的，并且也不会出现绝对值相同的两个元素
* 只要在任何一个车厢查到通行证就可以使火车通过，证明这种过程内在的逻辑是关系

分析上述信息，可知，若我们用一个量子比特的量子计算机，其中代表该位对应编号车厢下层放通行证，反之，代表该位对应编号下车厢上层放通行证，量子态的各个位数，从左到右从开始数，于是可以和从到节车厢对应，于是每个量子态都代表了一个排列的组合，这样的个量子态同时存在于量子系统中，它们的振幅决定了相应量子态的分布。

于是我们有量子态与解的映射关系：



把将量子态映射到实际解的过程称作解量子态，那么解量子态的程序如下：

|  |
| --- |
| *def* qb2sltn(*bstr*):  *#单个量子字符向量转化为对应的解向量*  *#返回一个列表格式的解*      nb=[*int*(i) for i in bstr]      nb=nb[::-1]      asltn=nb      for i,obj in enumerate(nb):          if obj==1:              asltn[i]=i+1          else :              if obj==0:                  asltn[i]=-i-1      return asltn |

而对于检查信息，定义第次检查第个车厢上的动作：



其中，分别代表 检查该车的上层，不检查，检查下层 通过此，我们可以将转化为，程序如下：

|  |
| --- |
| *def* pblmReceiver(*pblmlist*=[[2,2],[-1,2,0],[-1,0]]):  *#pblmlist: the given form of pblm*  *#numa,numb: the number of a,b*  *#check: for inspector i's j check, if the upper,1,downer,-1,dont,0*  *global* numa,numb      numa=pblmlist[0][0]      numb=pblmlist[0][1]  *global* check      check=np.zeros([numb,numa])  *global* oringinCheck      oringinCheck=[i[:len(i)-1:] for i in pblmlist[1:]]      for i in range(1,numb+1,1):          l=len(pblmlist[i])          for j in range(0,l,1):              v=pblmlist[i][j]              absv=np.abs(v)              if v!=0:                  check[i-1][absv-1]=  v//absv              else:                  break |

其中的 oringinCheck 是按照题目格式存储的检查信息。

e.g. 对于题给案例，它对应的矩阵为



# ZOC算法原理与实现

## 原理

给出两个基础量子比特，它们的线性组合：



可以表示该单比特量子系统中的任意量子态，将个这样的希尔伯特空间合并在一起可得：



其中任意一组基上的量子态分布由振幅：



所决定。

我们已知，可以通过构造振幅放大算子来增幅我们想要的基，在单比特情况下，它是一个旋转门：



由函数性质可知，如果对单比特线路应用这一量子门，则可以使量子态逆时针绕Y旋转，由定义在Bloch球上的量子比特表达式：



我们可以增加观测到的概率而减小观测到的概率，反之则可以增加观测到的概率，对应到问题分别是增加这一车厢放上层和放下层的权值。

我们做一种简单的处理，即认为若这一次检查了某一层，就增加一次那一层的权值，若这一次不检查这节车厢，就不改变车厢的权值，那么我们总共将有次旋转的判断，

* 显然，如果所有检察员都要检查这一层，那么解里只要有这一层就可以了，也就是含有这个比特的概率为，为此将每次的旋转量定义为。
* 如果某一个检察员要检查某几个车厢层，那么这一次构造的线路中就把相应量子比特的振幅增大，在初始振幅均等的情况下，这些车厢彼此间的竞争力不会有变化，也就是说，为了应付任一个检查员，选取任一个被检车厢的策略是同样好的。

至此，我们给出ZOC算法：

|  |
| --- |
| ***Input:*** |
| ***Output: result*** |
| 1.Allocate (a) qubits  2.Do  on the qubits  3.Do  on qubit j   1. If  , go to 3 2. If  , go to 3   4.Get the result, which contains the probabilities of all states |
| ***End*** |

## 实现

实现它的代码如下：

|  |
| --- |
| if \_\_name\_\_ == "\_\_main\_\_":  *global* sltn      sltn=[]      pblmReceiver()  *# extreme examples:*  *# [2,3],[0],[0],[0] : 所有组合都是解*  *# [2,3],[-1,-2,0],[1,2,0],[1,-2,0] : 没有解*        checkSpliter()      machine = pq.init\_quantum\_machine(pq.QMachineType.CPU)      qvec=pq.qAlloc\_many(numa)      cvec=pq.cAlloc\_many(numa)      prog=pq.create\_empty\_circuit()        dtheta=np.pi/(2\*numb)      for k in range(0,numa):              prog<<pq.RX(qvec[k],np.pi/2)          for i in range(0,numb):          for k in range(0,numa):              prog<<pq.RX(qvec[k],check[i][k]\*dtheta)        result = pq.prob\_run\_dict(prog, qvec, -1)      pq.destroy\_quantum\_machine(machine)        for key in result:          print(key+":"+*str*(result[key]))        resultAnalyzer(result)      print(sltn) |

运行算法之后，需要对得到的概率分布进行处理，处理的程序如下：

|  |
| --- |
| *def* resultAnalyzer(*result*):  *# 解的概率均匀分布，说明都是解，即有解*      prob=np.array(*list*(result.values()))  *#* *a值大时，各个概率值很小，这里把每个态的期望放大到 1*      prob=prob\*len(prob)      sltnstr=*list*(result)      if prob.std() <= 1e-6:          for i in range(0,len(sltnstr)):              sltnl=qb2sltn(sltnstr[i])              addsltn(sltnl)      else:          for i in range(0,len(sltnstr)):              if prob[i] >= 1e-6:                  sltnl=qb2sltn(sltnstr[i])                  addsltn(sltnl)  *# 解概率分布不均匀，取出较大概率的解检验，并存入sltn，如果实际上不是解，那么sltn还是空表* |

a值大时，各个概率值很小，这里把每个态的期望放大到1。

## 复杂度分析

量子计算的好处在于，可以通过适当的方法，把的for循环转化为，在这个问题里，式子



的计算，原本需要的时间，在量子计算机中却只需要一次运算，极大地提升了效率。

# 问题构造与检验

## 题给算例

题给算例下，输入为：

*pblmlist*=[[2,2],[-1,2,0],[-1,0]]

输出为：

|  |
| --- |
|  |

对应的量子线路为：

|  |
| --- |
|  |

## 算例2

输入：

pblmReceiver([[2,3],[0],[0],[0]])

显然，这个输入下，任意一种方式都是题目的解，输出为：

|  |
| --- |
|  |

对应的量子线路为：

|  |
| --- |
|  |

## 算例3

输入：

pblmReceiver([[2,3],[-1,0],[2,0],[1,0]])

输出：

|  |
| --- |
|  |

对应的量子线路为：

|  |
| --- |
|  |

# 改进思路

加入量子和传统电路的接口，在将输出存在经典寄存器中后，利用电压比较器等等构建交集电路，这样可以直接输出合适的解，不过这样的代价是需要过多的经典数字/模拟器件。