1 输入处理

问题输入的格式为：

$$pblmlist = [[a,b],Cinfo_{i=1}^{b}]$$

$$Cinfo_i = [Carr_{i,1}, Carr_{i,2}, ..., 0]$$

$$Carr[i,j] = \begin{cases} +j & \text{第}i\text{次检查会检查}j\text{车厢的上层} \\ -j & \text{第}i\text{次检查会检查}j\text{车厢的下层} \end{cases}$$

其中，$a,b,Cinfo_i,Carr_{i,j}$分别表示：

- 车厢数，
- 检查次数，
- 第i次检查的信息，
- 第i次检查车厢j的信息

从题目中可知：

- 通行证只有a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$Cinfo_i$，它的非零元素个数总是不大于a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OR关系

分析上述信息，可知，若我们用一个a量子比特的量子计算机，其中0代表该位对应编号车厢下层放通行证，反之，1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1开始数，于是可以和从1到a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2^a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：

$$|q_a \dots q_i \dots q_2 q_1\rangle \rightarrow [(-1)^{q_i} i]$$

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：

```python
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i次检查第j个车厢上的动作：

$$check_{i,j} \in \{-1,0,1\}$$

其中，$-1,0,1$ 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$Cinfo_{i=1}^{b}$转化为$Check_{b\times a}=\{-1,0,1\}^{3}$，程序如下：

```python
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

$$\begin{bmatrix} -1 & 1 \\ -1 & 0 \end{bmatrix}$$

2 ZOC 算法原理与实现

2.1原理

给出两个基础量子比特，它们的线性组合：

$$|\psi\rangle = \sin\theta|1\rangle + \cos\theta|0\rangle$$

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

$$|\psi_1\rangle \otimes \ldots |\psi_i\rangle \otimes \ldots \otimes |\psi_a\rangle$$

其中任意一组基上的量子态分布由振幅：

$$\alpha(|q_a \ldots q_1\rangle) = \prod_{i=1}^{a} \sin\left(\theta_i + \frac{\pi}{2}\bar{q_i}\right)$$

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：

$$Q = Y_{4\theta} = \begin{bmatrix} \cos(2\theta) & -\sin(2\theta) \\ \sin(2\theta) & \cos(2\theta) \end{bmatrix}$$

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：

$$|\psi\rangle = e^{i\gamma}\left(\cos\frac{\theta}{2}|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}|1\rangle\right)$$

我们可以增加观测到$|1\rangle$的概率而减小观测到$|0\rangle$的概率，反之则可以增加观测到$|0\rangle$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b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1，为此将每次的旋转量定义为 $\Delta\theta = \frac{\pi}{2\times b}$。
- 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: $Check_{b\times a} = \{-1,0,1\}^3$

Output: result

1.Allocate (a) qubits

2.Do $Y_{\frac{\pi}{2}}$ on the qubits

3.Do $Y_{Check_{i=1,j=1} * \Delta\theta}$ on qubit j

1. If $j \leqslant a$, go to 3
2. If $i \leqslant b$, go to 3

4.Get the result, which contains the probabilities of all states

End

2.2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。

2.3复杂度分析

量子计算的好处在于，可以通过适当的方法，把$\mathcal{O}(N)$的 for 循环转化为$\mathcal{O}(1)$，在这个问题里，式子

$$\alpha(|q_a...q_1\rangle)=\prod_{i=1}^{a}\sin\left(\theta_i+\frac{\pi}{2}\bar{q}_i\right)$$

的计算，原本需要$\mathcal{O}(N2^N)$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。

3 问题构造与检验

3.1题给算例

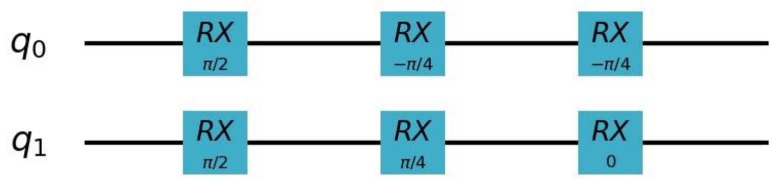
题给算例下，输入为：

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

输出为：

```
00:0.14644660940672624
01:7.703719777548943e-34
10:0.8535533905932742
11:0.0
[[-1, -2], [-1, 2]]
```

对应的量子线路为：



3.2算例 2

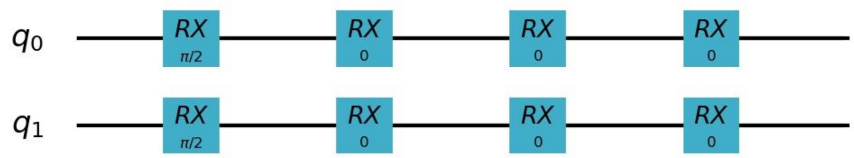
输入：

```
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：

```
cripts/quantum/zoc_solver.py"
00:0.2500000000000001
01:0.2500000000000001
10:0.2500000000000001
11:0.2500000000000001
[[-1, -2], [1, -2], [-1, 2], [1, 2]]
```

对应的量子线路为：



3.3算例 3

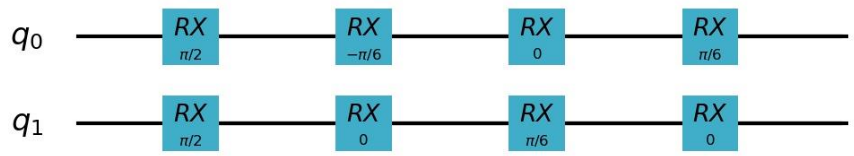
输入：

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

输出：

```
00:0.12500000000000006
01:0.12500000000000008
10:0.37500000000000033
11:0.3750000000000002
[]
```

对应的量子线路为：



4 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。