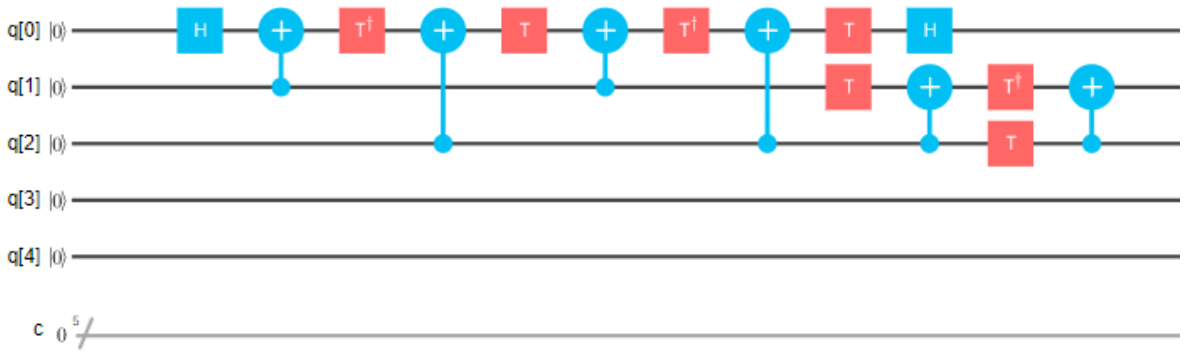
Calcul Quantique IBM-q practice : Algorithme de Grover

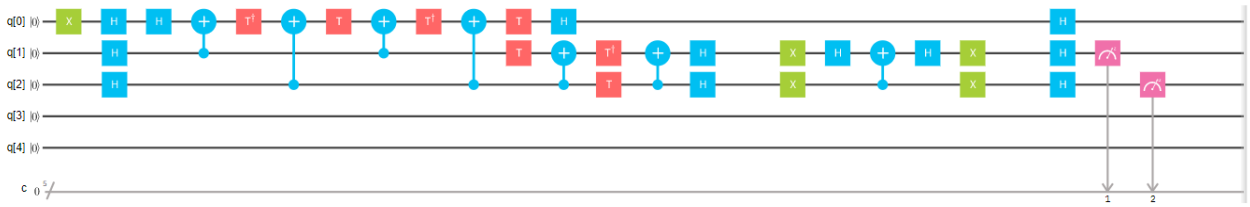
Circuit pour N = 4

Il faut tout d'abord réaliser l'oracle. Bien sûr, si on réalise l'oracle à la main spécifiquement pour marquer une entrée, la sortie de l'algorithme ne nous apprend rien. Mais dans de vrai application, l'implémentation de l'oracle pourrait être inconnu, ou implémenter n'importe quelle formule booléenne dont il peut-être très difficile de trouver une solution.

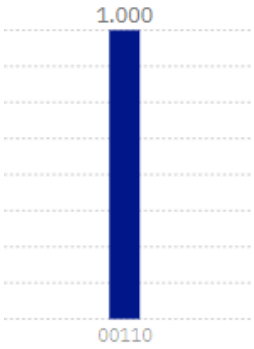


Une porte de Toffoli, ou CCNOT, qui change le premier Qbit quand les deux autres ont la valeur 1

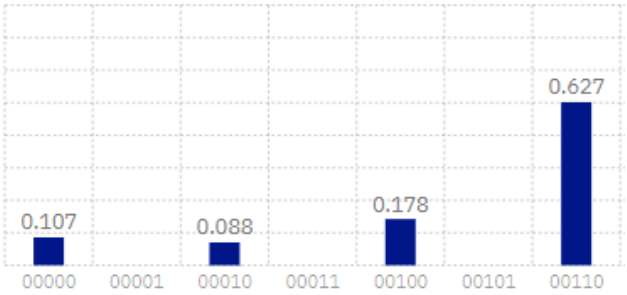
Le circuit dans son ensemble :



Le résultat de la simulation :



Et celui de l'expérience :



On trouve bien la sortie 11 comme on le voudrait, qui indique que la fonction f de notre oracle U_f en haut est bien f(1,1) = 1

On essaie maintenant de l'implémenter pour 4 qbits, avec N=16

Cette fois, on va utiliser qiskit pour créer le circuit. Tout d'abord, observons que le dernier qbit n'est pas nécéssaire. En effet, on a pu voir dans les exercices que le dernier qbit se retrouve désintriqué des autres à la fin. En fait, commencer avec XH devant le qbit "cible" fait que après l'oracle, le qbit cible est désintriqué et la phase d'un certain résultat est inversée. Mais il est plus efficace d'implémenter directement ce changement de phase, et de n'utiliser que 4 qbits au lieu de 5.

In [36]:
```
import qiskit as qk
from qiskit import BasicAer, execute
from qiskit.tools.visualization import plot_histogram


from math import pi
```

On va créer une fonction qui applique notre oracle U_f aux qbits donnés, et une autre qui se charge de l'amplification.

In [125]:
```
class CustomQuantumCircuit(qk.QuantumCircuit) :
    def Uf(self, qr0, qr1, qr2, qr3, target): #marque par une phase "-" l'entré
e qu'on veut marquer
        if target[3] == "0":
            self.x(qr0)
        if target[2] == "0":
            self.x(qr1)
        if target[1] == "0":
            self.x(qr2)
        if target[0] == "0":
            self.x(qr3)

        self.cccz(qr0,qr1,qr2,qr3)

        if target[3] == "0":
            self.x(qr0)
        if target[2] == "0":
            self.x(qr1)
        if target[1] == "0":
            self.x(qr2)
        if target[0] == "0":
            self.x(qr3)

    def cccz(self, qr0, qr1, qr2, qr3) :
        self.cu1(pi/4, qr0, qr3)
        self.cx(qr0, qr1)
        self.cu1(-pi/4, qr1, qr3)
        self.cx(qr0, qr1)
        self.cu1(pi/4, qr1, qr3)
        self.cx(qr1, qr2)
        self.cu1(-pi/4, qr2, qr3)
        self.cx(qr0, qr2)
        self.cu1(pi/4, qr2, qr3)
        self.cx(qr1, qr2)
        self.cu1(-pi/4, qr2, qr3)
        self.cx(qr0, qr2)
        self.cu1(pi/4, qr2, qr3)

    def amp(self, qr0, qr1, qr2, qr3): #change la phase de |0000>

        self.x(qr0)
        self.x(qr1)
        self.x(qr2)
        self.x(qr3)
        self.cccz(qr0,qr1,qr2,qr3)
        self.x(qr0)
        self.x(qr1)
        self.x(qr2)
        self.x(qr3)


def init():
    qr = qk.QuantumRegister(4)
    cr = qk.ClassicalRegister(4)
    qc = CustomQuantumCircuit(qr, cr)
    return qr, cr, qc
```

Maintenant il faut effectuer l'opération un certain nombre de fois. Calculons t (l'angle teta).

sin(t) = sqrt(MN) = sqrt(1/16) = 1/4

teta ~ 0.25 ~ pi/2 * 1/6

Cette fois nous ne pouvons pas arriver à un résultat exacte. pour N=4 nous avions teta ~ pi/2 *1/3 et donc ajouter 2* teta nous permettait d'arriver pile à pi/2.

Ici nous pouvons arriver à 5pi/12 (ou 7pi/12).

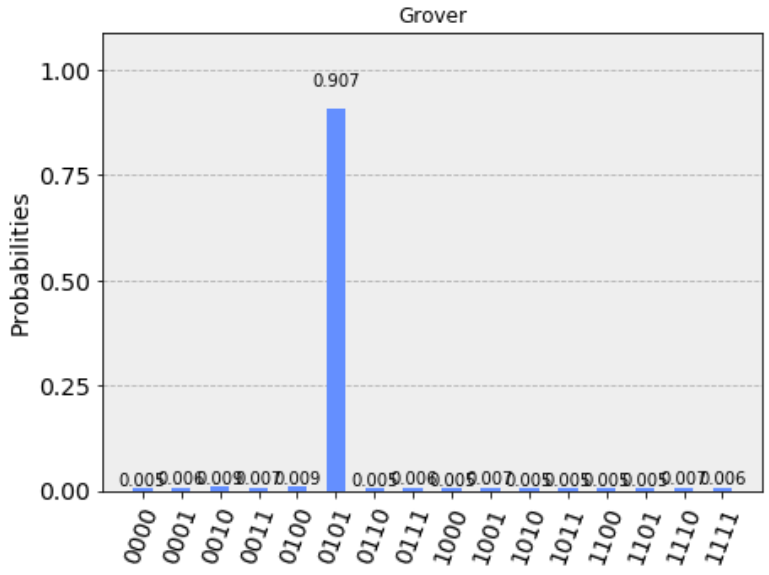
In [126]:
```
qr, cr, qc = init()

target = "0101"
qc.h(qr[0])
qc.h(qr[1])
qc.h(qr[2])
qc.h(qr[3])
for i in range(2): #ou 3
    qc.Uf(qr[0], qr[1], qr[2], qr[3], target)
    qc.h(qr[0])
    qc.h(qr[1])
    qc.h(qr[2])
    qc.h(qr[3])
    qc.amp(qr[0], qr[1], qr[2], qr[3])
    qc.h(qr[0])
    qc.h(qr[1])
    qc.h(qr[2])
    qc.h(qr[3])

qc.barrier(qr)
qc.measure(qr[0], cr[0])
qc.measure(qr[1], cr[1])
qc.measure(qr[2], cr[2])
qc.measure(qr[3], cr[3])

simulator = BasicAer.get_backend('qasm_simulator')
result = execute(qc, simulator, shots = 4096).result()
counts = result.get_counts(qc)
plot_histogram(counts, title='Grover')
```

Out[126]:



On observe qu'on obtient bien la sortie de f la majeure partie du temps. Les résultats sont légèrement meilleurs avec 3 itérations qu'avec 2, car on se retrouve légèremenr plus proche de pi/2, mais sur un vrai circuit quantique il vaut sûremenr mieux utiliser deux itérations pour minimiser le nombre de portes, et la perte de cohérence du circuit.

In [ ]: