PHYS 344 Final Project Report

December 21, 2022

Quantum Computing - Grover's Algorithm

Aidan McClung

What quantum computing is not

There's a very good chance that you have heard of quantum computing, at least in some capacity. Quantum computing is something that became very popular and known about in the public eye, but it was very misrepresented. As such, I want to make sure I take the time and really walk through what it is and make sure we're all on the same page.

Quantum computers are often introduced with an initial comparison to a classical computer. The classical version, at it's most fundamental level, these things that are either zero or one. In contrast, a quantum computer uses something which may be zero, or one, or something[1] in between the two. With this fundamental expression, that can be any combination of zero or one, we can use it in cool ways. If we were to combine them into a piece of data for example, like a string of characters, this combination could express every possible combination at the same time. Then, if we were to do some operation on this combination, the result we would get would be itself a combination of all possible results. Thus we can do in one single step, that which we would have had to do many times over in the classical situation.

This is where those explaining these concepts usually stop, and it's fairly clear why. Thus far we've concisely explained our subject in a way that is understandable to most people, and, perhaps more importantly, nothing in the above explanation is incorrect or explained in such a way that it simplified beyond the point where it still represents what its describing. More importantly though, we have just presented a fantastic concept for a very powerful new type of computation, and those people we're talking to are excitedly thinking about the possibilities already.

This halting has led to the general idea being conveyed and remembered that quantum computers are these new type and genre of computer, that are way better and light years ahead of the competition from classical computers. The main thought is that they can do a practically infinite number of things in one go.This is where there's a bit more to the story.

While it's entirely true that quantum computers have the ability to do one operation and get the result for a range of inputs, there's a catch. The answer we get is a "superposition" of all the answers (people like to throw around the word superposition without really understanding what it means). We have an answer, which is somehow a combination of every possible answer. The problem, is that, while the specific answer we're after (like the password we're trying to guess) is in that superposition, when we try to extract it we will get only ONE of the possible answers, usually with an even probability for each of the possible answers. So, while you may be able to compute all 1 million possible passwords in one go, you can only get out one random password.

This is exactly the same principle behind Schrodinger's cat. While inside the box, the cat can be thought of as being both dead and alive. In the same way, while inside the "magic little box" that is our quantum computer, our result might be both right and wrong. When we try to look at it though, our result will lock in one of the options, the same way that the cat will be either dead or alive, but not both, once observed.

This sounds much less impressive, and I entirely understand that if I was trying to inspire others with what I was doing, or convince investors it was worthwhile, I would definitely not

[1]This tripped up the author for many years. The "something" in this common definition is much more so a "anything" than a something. In a more formal sense, the fundamental element of a quantum computer is any linear combination of 0 or 1, where the global phase has been dropped.

make it sound like I was making a thing which could just guess at answers.

Thankfully, there's still more to the story. Quantum computationalists main purpose is to devise ways to harness quantum computers. Most commonly, this means that they are trying to create some system that will skew the odds in their favour. If they can skew the odds enough, they can maybe even guarantee they get the answer they want, in which case, they would have a magic box. In reality though, there will always be some possibility of getting the wrong answer, even in classical, deterministic, computing[2]. Thus the task is to minimize the chances of getting the wrong answer while maximizing the chances of getting the right one.

Thus far I have avoided explaining how a quantum computer does actually work, and have tried to be vague and general, and paraphrase others. There are some very interesting ways that researchers have come up with to do this, and they vary in some fundamental ways. Before mentioning them however, I want to go through some of the basics of what's involved in quantum computing, and really explain it properly on my own.

What is (the most basic intro to) quantum computing

One of the first things we encountered during our journey through quantum mechanics this semester was the idea of a particle in a box.

We started by splitting the box in two, and we said that our particle could be in one of two states: $|L\rangle$, representing when it was in the left side of the box, and $|R\rangle$, representing it's presence in the right half of the box. These two states were necessarily entirely distinct; a particle that was $|L\rangle$ could not be, in any way, $|R\rangle$. The same holds for all states, they are entirely distinct things.

Now, it is very reasonable to ask; if we were able to build such a box and put a particle in it, could we rename $|L\rangle$ and $|R\rangle$ to $|0\rangle$ and $|1\rangle$ and thus make a computer? Absolutely we could. The downside however would be that we just spend a lot of time trying to get a particle into a box, and we know have something that is an exceptionally bad classical computer. It has only 0 and 1, and none of the new special things that quantum computers are heralded with. Evidently, we need to go a bit further into quantum mechanics to get the kind of thing we're after.

We next introduced the idea that maybe the particle was moving, or we didn't know where it was at every possible moment. We brought in a fundamentally important concept to represent this state of not knowing; the general state. We said that the particle was absolutely in the state $|\psi\rangle$, where $|\psi\rangle$ was some combination of every state that we could have, ie.

$$|\psi\rangle = \alpha\,|L\rangle + \beta\,|R\rangle \tag{1}$$

Where α and β are constants, that represented the relative "amount" of their state that was part of the superposition (sum) of states $|\psi\rangle$. As an example, we thought of this "amount" as being the amount of time that a moving particle spent on either side of the box. Equivalently, this could be the potion of the total volume each segment comprised, or the likelihood, if we

[2]At any moment a random thing could pop in from space and yoink out an electron from your computer, ruining everything. Thankfully, this is fairly uncommon, and we also have done a lot of work to develop ways to negate these kind of errors, but this is not really the place for that.

were to take a snapshot of the inside of the box at any given time, that the particle would be in that segment.

This is MUCH MORE promising for building a differenter computer out of. Instead of having a state that is only 0 or 1, we now have a state which is some combination of $|0\rangle$ and $|1\rangle$. In fact, this is exactly what we did. We will not talk about, in this paper, the actual ways that people have implemented this. As theorists, we're just going to make up the math, and state that we are assuming that there's a sufficiently talented experimentalist out there in the world somewhere who can build the system we design (or, more accurately, don't design).

On the basis of twos

I would also like to quickly mention another thing we will not address here. The next thing we moved to after generalizing our left and right states was that there was nothing inherent about the left and right involved in the definition we had made. Specifically, we were able to add on more "halves" on the outside of this existing box, or split the box up more, and get a lot more than 2 possible states. This is certainly *possible* in quantum computing as well, but we will pretend it's not for the sake of maintaining our sanity as we learn about it, using just two possible things.

Back to computing though. In classical deterministic computers, we have this thing that is 0 or 1. We call this thing a "bit". The name comes from a contraction of "BInary information digiT", which is commonly attributed as being first used by John W. Tukey of Bell Labs, in January of 1947.

In quantum computing we use something that we call a "qubit", for QUantum BIT. It is quite simply just a two dimensional vector, denoted by a two column vector of the coefficients which represent the vector (or state) relative to whatever basis we're using.

The most common of these bases is the "computational basis". It's basis vectors are $\begin{bmatrix}1, 0\end{bmatrix}$, which we denote $|0\rangle$, and $\begin{bmatrix}0, 1\end{bmatrix}$, denoted by $|1\rangle$. Importantly, remember that we still have a general state, or qubit, $|\psi\rangle$, which we could represent as $\alpha\,|0\rangle + \beta\,|1\rangle$, or, equivalently now, $\begin{bmatrix}\alpha, \beta\end{bmatrix}$.

There is another significant basis for our uses that I think is important to bring into the mix at this point. The idea behind it follows an idea we already encountered in out quantum mechanical journey this semester. When working with two particles, and trying to determine how the system changed with time, we found it was significantly easier and more beneficial to look at the sum and difference of the two momentums, p_+ and p instead of p_1 and p_2. In the same way, we have a basis in quantum computing consisting of two normalized vectors:

$$|+\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + |1\rangle\right)$$
$$|-\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle - |1\rangle\right)$$

This basis involves an even superposition of the computational basis, where each option is equally likely. Predictably, we will eventually need more than a single qubit, and, when we get to that point, we will start "stacking" qubits together; in reality we're just using multiple, but stacking sounds more fun.

But how do we DOOOOOO things?

Thus far we have defined qubits, that are instance of states. Right now though, they are quite boring. We can summon them into existence by defining them, but we have nothing we can do with them; they just sit there like that forever[3]. We need some way of being able to do something with our states, and for that, just like we did in class, we will bring in *operators*. Once again, these operators which act on our states, represented by vectors, are represented by matrices[4]. In Quantum computing, we call the implementation of an operator a "gate", and the main thing people do is try and come up with combinations of gates that actually do things. We must always meet the condition that the sum of the probability for each possible thing is overall equal to 1[5]. In order to maintain this condition, we only work with unitary operators; we can shift things around inside the relative chances and amounts for each component, but we can't duplicate or clone anything[6,7]. They are called gates simply because they are the analog to logic gates in deterministic computing, and we compare those two more often then we would want to talk about matrices. As the analog, there are equivalent quantum gates for the logic gates. To provide an example, the matrix (and thus gate) $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ will have the same effect as a NOT gate; it will turn a $|0\rangle$ into a $|1\rangle$ and a $|1\rangle$ into a $|0\rangle$. There are many gates, and they can be combined to make more complicated operations. We will not talk about most of them, but the primary role of those working with quantum computers is to create these combinations.

On Quantum Algorithms

It turns out that this task of combing gates to actually do things is exceptionally hard. At least, it is to the author, who, in the time spent researching and writing this paper, did not invent even a single new quantum algorithm.

There are however, a few quantum algorithms that are important and deserve special mention, as well as a few things that can be said of quantum algorithms in general.

Firstly, it bears emphasizing that any classical, deterministic, algorithm can be implemented on a quantum computer. At this point in time quantum computers are expensive

[3]A very prevalent problem for the experimentalists to figure out is that qubits most certainly do not just sit there forever. Certainly not in any way that has been devised to implement them thus. Thus a common goal is to try and maximize the time that qubits will stick around. This is one of the key metrics 'media' uses to compare quantum computers, and I've forgotten the name and can't be bothered to go source it for a footnote on a joke.

[4]It turns out that quantum computing, just like quantum mechanics, is actually just a whole lot of linear algebra in disguise.

[5]I think, at some point during the drafting process, I cut out the point where I mention this earlier. Thus this is a little bit harsely introduced, and I'd like to apologize to any readers who would not see this statement as trivial and thus appreciate a gentler introduction prior to this point.

[6]The subject of cloning and duplication is one that is a fundamental concept and usually introduced before the things this paper is working towards. However, I know many of my colleagues are writing about this process in depth, so I would encourage any readers to detour to their work if they wish to learn about the subject

[7]This page is getting a lot of footnotes… It would be especially tragic if the footnotes don't transfer into LaTeX well… (note the author has stubbornly insisted on typing up the entire report in the language of LaTeX, entirely within a google doc.)

and specialized, so it is never beneficial to do so, but it is possible. This possibility is fairly easily explained; if we simply only let our qubits be $|0\rangle$ or $|1\rangle$, we have exactly the same machine as any other computer.

So, while "quantum algorithm" technically refers to any algorithm that can be run on a quantum computer, it is generally reserved for those special cases that are either impossible on a classical computer, or do the same task significantly faster[8].

As was mentioned in the introduction, when we do operations on a superposition of many things, we will get back a superposition of the results of doing that operation on the many things. With the knowledge that we've once again found linear algebra in disguise, it's easy to see the connection to vectors, vector components, and matrix multiplication. The result we obtain after a basic operation, is generally evenly distributed across the possible options. In a more formal sense, each possible result has a particular weight attached to it, and we're not at all guaranteed to be able to successfully extract a specific result from the superposition. What most quantum algorithms have in common is that they each use clever tricks and strategies to maximize the possibility of getting the result we're after. Without going into the details of how, the following are some of the most commonly known "quantum algorithms":

Shor's algorithm provides a way to factor very large numbers fairly quickly, and is the most famous quantum algorithm. It is of particular concern to cryptography, since the entirety of our digital security is build around the idea that it is very hard (and thus impossible in practice) to look at a large number and know what factors it has, but very easy to look at the factors and verify they produce the large number in question.

The Quantum Fourier Transform does exactly what it sounds like it does. The QFT is even faster than the Fast Fourier Transform, which has exciting implications since the FFT widely hailed as the most influential algorithm on society as a whole of all time.

Lastly, Grover's algorithm is a way to search through an unsorted collection of items (and pick out a target one) faster than a linear search; the best classical way to do so. Grover's algorithm is what we're going to talk about, but we will introduce it in an much easier form; the contexts of passwords, and picking out the correct one from all possibilities.

Roleplaying

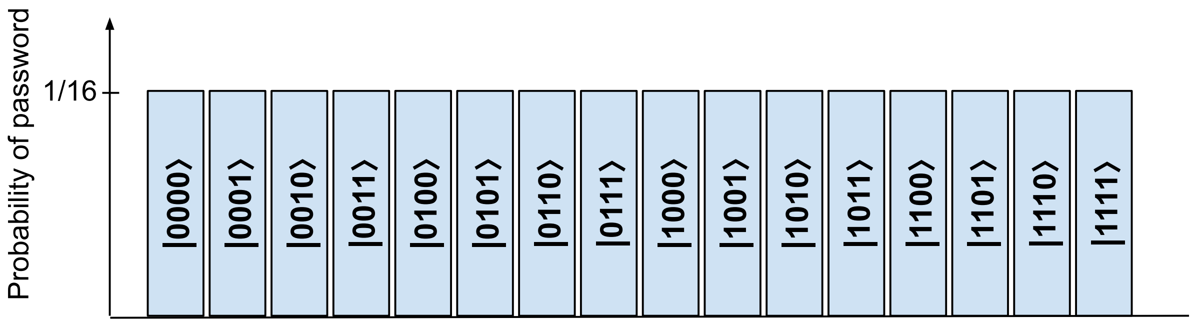
You had decided to be better about your digital security, and were changing your passwords to be different on every site. You did however, anticipate you not being able to remember what they each were, and so you built a mechanism to help you guess right if you did forget. This mechanism was a very secure backdoor, probably better than your original password, since even fewer people would know the specific way this device could make it very easy to figure out what your password was. This device, this mechanism, was a gate that could be used in a quantum computer, and it does a very specific thing. It operates only on states representing potential passwords, and it does absolutely nothing to most of those states, but it will flip the sign (of the amplitude) of the state corresponding to the correct password. This may seem like it will just show a potential intruder what your password is, but you had

[8]I'd like to note here that in the context of algorithms and computation, "faster" exclusively means "less operations required" and has no indication of the physical time it takes to run an algorithm on a specific implementation of a computer.

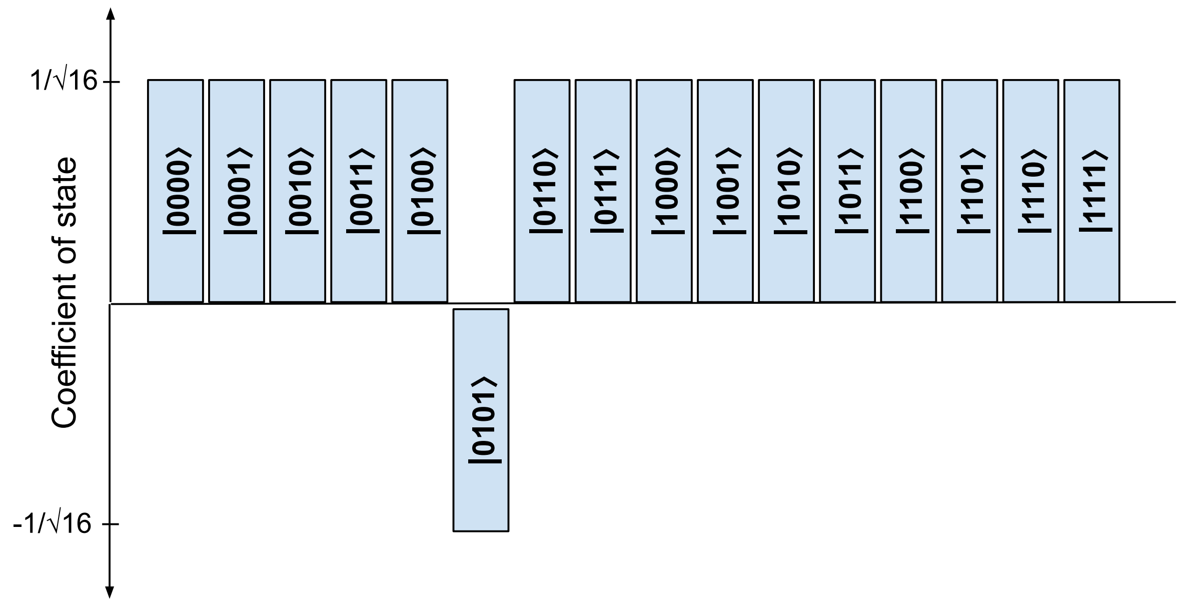
remembered what you learned, and that the probability of getting a state when we extract an answer from a quantum computer is the square of the amplitude, and thus, changing the sign wont matter because it will be squared anyway. This quantum gate is just the implementation of the textbox where you type in your password and click enter, so it's not really any less secure to leave it lying around than it would be to leave your computer lying around.

Now, a lot of time has passed, and you have an assignment you need to hand in, but you've forgotten your login information. You know that you made it easy on yourself, specifically to handle the possibility that you had to explain to anyone how to figure out your password, and make it fairly easy to explain. Your password is a string of 0s and 1s. You learned in your CISC 203 class, that when dealing with passwords, there are (number of possibilities for each character) to the power of (the number of characters) possible passwords. As well, you know that if the item or password you're looking for is randomly distributed throughout the collection of options, it will, on average, take half the number of possible passwords to guess it. This estimate is based on the most efficient deterministic method for handling this scenario; making a list of all the options and then plugging them in one at a time. However, you wanted to do better, which is why you built this fancy box for yourself to use.

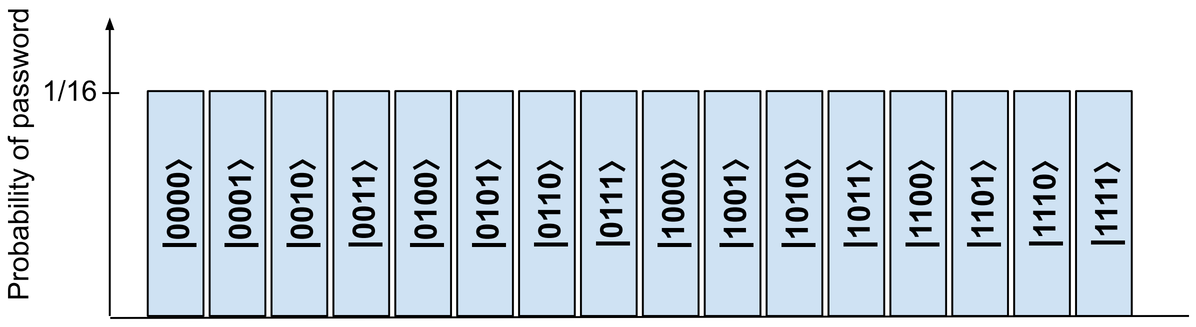
Anyway, back to figuring out your password instead of being distracted by thinking about how much better than everyone else you are. Since your password is a string of 0s and 1s, you can just build a quantum circuit with the same number of qubits as you have characters in your password. You really were doubling down on the "being able to teach others" angle, and so your password is only four characters long. You have no memory whatsoever, so you might as well set it up with each qubit having an equal distribution between $|0\rangle$ and $|1\rangle$, also known as $|+\rangle$. Because each of your four qubits is equally likely to be $|0\rangle$ or $|1\rangle$, the whole circuit has an equal coefficient on each possible state, or password. This could be helpfully visualized with the following figure that you made, which is also to make up for the fact that you hadn't explained any of the notation for combining qubits.



Now, you have your special box. It was given a name, but that was by the math people, so you'll leave that for the math section. This box will flip the sign of the solution state. So if we give it a state which is an even superposition of all states (as in, that which is visualized by the figure above) we'd have the following thing happen:



To reiterate once again, when we take an option out, we have the following probabilities:



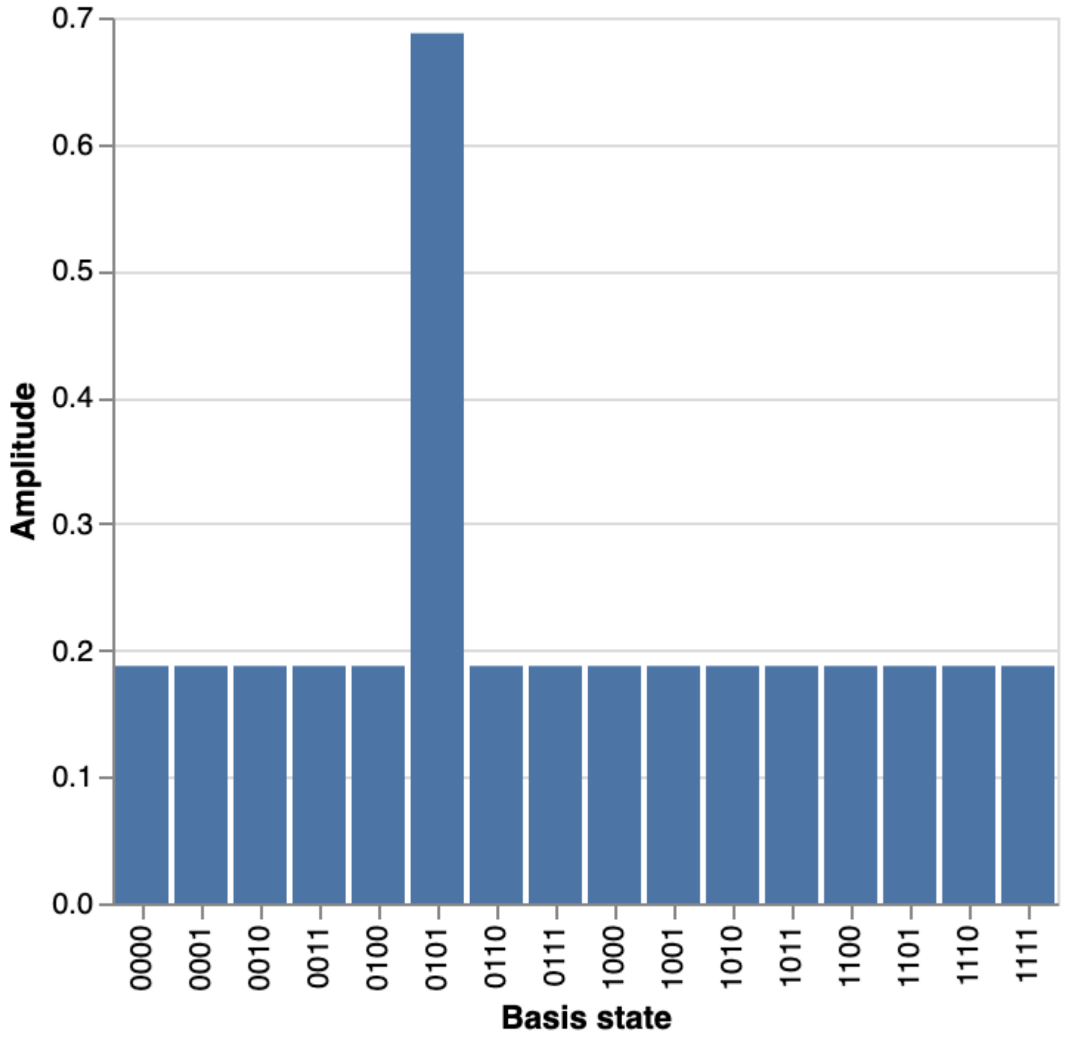
So we need to do something with the sate we have. (Also as a note, you realize that anyone looking at the image might think "well, you're an idiot, the password is obviously "0101", it's flipped." And to that you'd simply point out there's no actual way to see the image with the flipped sign, since, in order to build such an image, you'd need to take the result a LOT (more than 16) times and count how many times you get each option.

So now you need to be smart, and find some way to get out the right option. The best way[9] to do this would be if we could do something, and make the probability of our password

[9]There is a not-best way to do this that is generally introduced first, that involves testing pairs to get a slight improvement, but you're skipping over that for the sake of time.

higher and higher, while making every wrong option lower and lower. This turns out to be something that you can do, but you should really leave explaining how to your math friends, since the process involves a lot of inner products, and is also really helpfully explained as a next step after making the “flip sign” gate that you skipped over.

Essentially though, the idea would be, as you demonstrate on your quantum computer simulation built using python code using the PennyLane package from Xanadu, to make the probabilities more like this[10]:



[10]The image generated was actually the result from the codebook module where I had applied the diffusion and oracle operator once each, but those words do not have meaning at this point, though I figured I’d contextualize it.

On the Linear Algebra of the Grover Operator

The title here mentions a Grover operator, and maybe that's too much of a spoiler and I shouldn't have mentioned it. Regardless, it's too late now, and so yes, we will eventually make something called the "Grover Operator" and yes, "Grover's algorithm" is in fact:

1. Apply the Grover operator until the probability of the state you want is maximal.

This final result is a bit anti-climactic, but the operator bundles together a lot of interesting things, and so the journey is worth taking.

To begin with, we'll need to define a few things. We will use $|s\rangle$ to refer to our solution state, the state we're after. We will be operating in some space, which is technically a tensor product of qubit basis states, but is sufficiently thought of as a space with $N = 2^{(\text{number of qubits})}$ basis states. We will use the state $|\psi\rangle$ to refer to the state that is the even distribution of all the N basis states. With that out of the way, we will follow the same general idea and steps as our forgetful friend did when she explained they explained things, but from the math side. The first thing we want to do, while we know what $|s\rangle$ is, is to construct the box they mentioned. This was called a lot of things, but importantly, it was a quantum gate, which means, for the math, it's just a matrix, or an operator in the general sense. We wanted this operator to have the action that it would do nothing for anything that wasn't $|s\rangle$, and it would flip the sign of the coefficient on $|s\rangle$. A good place to start building this is the "do nothing" part. We already have covered the "do nothing" operator- that's the identity operator I_N. So now we need to do something only when we get $|s\rangle$. We would first need to check whether something was $|s\rangle$, and thankfully, we had some notion of a vector operation that measured the "amount of similarity" between two things- the inner product. So $\langle s|\,|s\rangle$ would be 1 and $\langle s|\,|\text{anything else}\rangle$ would be 0. Thus, as an initial idea, subtracting the amount of "s-ness" in any input from the identity would give us back exactly the same thing but with no $|s\rangle$ component.

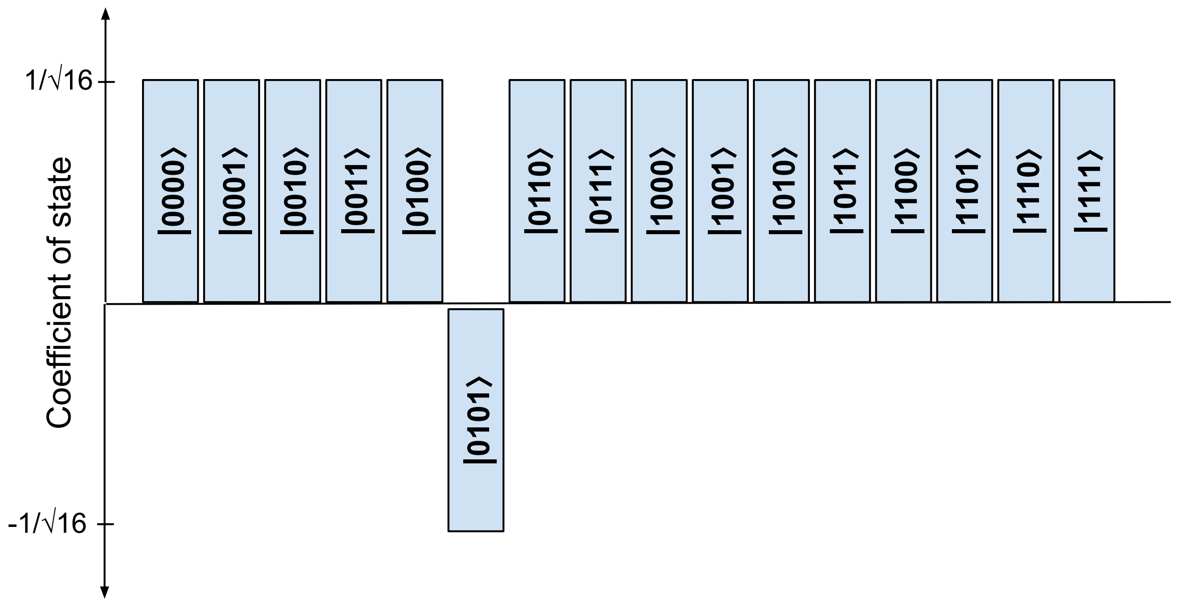
$$I_N - |s\rangle\langle s| \tag{2}$$

This isn't exactly the action we were after, but, if we simply subtract the $|s\rangle$-ness twice we would have negative the amount of $|s\rangle$ness we did originally, and once we do this, we have what we call the "Oracle" operator, since it will basically tell us if we're right or not. It, and an example of it being applied follows:

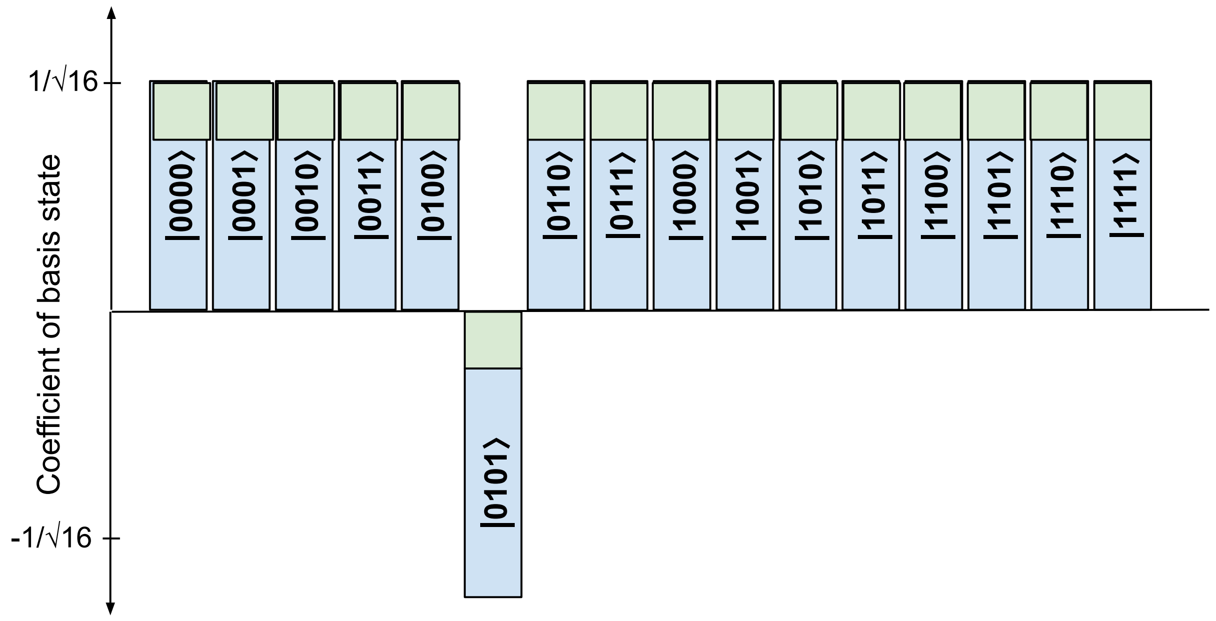
$$U_f = I_N - 2|s\rangle\langle s| \tag{3}$$

$$U_f|\psi\rangle = I_N|\psi\rangle - 2|s\rangle\langle s|\,|\psi\rangle \tag{4}$$

This works exactly as we wanted and expect, and will do the following to $|\psi\rangle$



Now the next thing we wanted to do was to be able to steal some of the amplitude from everything that wasn't the solution, and give it to the solution component. One way to do this would be to have a very similar operator as the oracle, which would take the amount of "everything-ness"of the state it was given times the "everything" state, and subtracted the input state. We'd want to take two times the "everything-ness" as before, so that we don't get zero when given the state $|\psi\rangle$. Why this would do what we want isn't super clear, and bears clarifying why it would work. We will end up having a negative portion of all the things we put in, and then subtracting them, but, if we already have some portion of it that is negative, subtracting from that will make it more negative, which would make it more likely:



Algebraically, this operator is something we call the "Diffusion operator" since is spreads out and diffuses the amplitudes.

$$D = 2\left|\psi\right\rangle\left\langle\psi\right| - I_N \tag{5}$$

The diffusion and oracle together form the "Grover operator", and, when we apply it, the probability of our solution state will increase by a bit. This result of doing this with a simulation of a quantum computer is shown as the final item in the previous section, as is mentioned in the attached footnote there, should any readers be interested in going back. The amount the amplitude of the solution component increases is based on how much of the even superposition state your solution state is because of the inner product. However, since it's an even superposition, this amount is actually roughly $\sqrt{N}$. You'll have to apply the Grover operator a certain number of times in order to maximize the amplitude of $|s\rangle$, and that number isn't exactly a super easy number to know, and the author was forced to determine it empiracally, and does not fully understand what she did in order to do so and will thus leave the explanation alone.

In Conclusion

There were a lot of things that were covered in this paper. A significant amount of the things I mentioned were more general background things, and, given that the specifics of the project were to work through a specific quantum algorithm, they were maybe unnecessary.

The specific algorithm we talked about was Grover's algorithm. The idea was that, using inner products, the identity operator, and subtraction, we could create quantum gates which would amplify the coefficient of target item in an even superposition. This is most generally a way to search for a particular item in an unsorted collection of those items. We only explicitly covered the case in which the items are evenly distributed, but, since we can repeat the process, it should be clear that it is possible for the amplitudes to not be evenly distributed to begin with. The algorithm itself is the particular mechanism behind the commonly cited example application of a quantum computer, where it tries to input every single possible password at the same time, and then, since it tried all of them, one of them will be correct. As the majority of the report spent the time going over, quantum computers are not actually the almost magical technological leap that they are often made out to be, and the catch is that they wouldn't be able to immediately input every possible option. Instead, we have to do more operations in order to try and artificially enhance the chance that the output we get is the one we want.