Reading

Lectures this week are picking and choosing material from chapters 1–7 of the quantum textbook. You should at-least-skim-read all of them, but here are the particular sections being emphasized: sections 3.2–3.4, 4.3–4.4, 5.1–5.3, 6.6, and all of chapter 7. Chapter 7 has aspects of being a recap, and those aspects are sprinkled through my lectures. My lecture notes also include material from the textbook’s second edition, which doesn’t exist yet—except for those notes. The course will finish by covering chapters 8–10 in pretty much linear order.

The **Final Exam** will stay at the appointed Monday 12/10 3:30–6:30pm time. It will be a three-hour cumulative exam.

Assignment 7—the last

(1) Consider graphs $G = (V, E)$ in which every node $u$ has an integer “capacity” $c_u$. To make this vivid, let us consider the nodes $u$ to be medieval towns, edges to be roads between towns, and $c_u$ to be the number of roads into town $u$ that an invader needs to control in order to take over the town. King Arthur and his band of $k−1$ other knights start in town $s$. There is a target town $t$ and Arthur wins if he enters the town $t$. The final rules are:

- The knights do not have to follow roads—they may ride “in country”—but (intuitively for supply reasons) they must occupy $c_u$ neighboring towns in order to conquer town $u$.
- If all knights leave a town $u$, then $u$ needs to be re-conquered if a knight wishes to occupy it again.
- More than one knight may occupy a town.

For an example, consider the following graph with $k = 2$ knights initially on the two nodes at left:
They can conquer the two nodes at right, in particular because the town of capacity 1 can be freely entered from any neighboring town. (A knight riding “in country” would, however, need help from a knight in a neighboring town to enter it.) Observe, however, that two knights starting at the two nodes at right cannot conquer the two at left. This is not because of the directed edges—the graph as above is undirected. (The edges show as directed because I used the online “Finite State Machine Designer” to draw it—and because all this gives a hint for part (b) below.) Instead it is because of the different capacities of the two nodes in the middle.

For (a) and (b) we consider $k$ to be a fixed number and define $L_k$ to be the language of the following decision problem:

**Instance:** $G = (V,E)$ with distinguished nodes $s,t$ and capacity function $c : V \rightarrow \mathbb{N}$.

**Question:** Can $k$ knights starting on $s$ conquer node $t$?

(a) Show that each $L_k$ belongs to NL.

(b) Show that $L_2$ is NL-complete by $\leq_{m}^{\log}$-reducing the Graph Accessibility Problem (GAP) to it.

(c) Now suppose $k$ is allowed to vary—that is, that $k$ too is part of the input. Call the problem or its language $L_{\infty}$. Try as best you can to answer these two questions and justify your answers:

- Is $L_{\infty}$ in NP?
- Is $L_{\infty}$ in DLBA—that is, in deterministic linear space?

**Notes:** I have on-purpose left some details of (c) vague so that different answers can receive full credit if the justifications are appropriate enough. On (b), if the graphs were allowed to be directed then $L_1$ with all node capacities 1 would simply be the GAP problem. But for an undirected graph the $L_1$ language does belong to deterministic logspace (a very deep result proved by Omer Reingold in 2003, actually). So you have to use $L_2$ and take the above example as a hint of how to make an undirected graph “behave like” the directed one given as the argument of your reduction from GAP. (12 + 18 + 12 = 42 pts. total)

**Answers:**

(a) We need only about $\log(n)$ bits to maintain the position of any one rider, so if the number $k$ of riders is fixed then the total number of bits $k \log(n)$ is $O(\log n)$. To guess and verify a solution to the conquest problem, we only need to maintain the current position of each rider and verify that enough riders were positioned on adjacent nodes each time a node is conquered. We do need to guess a good next-move for each knight, so this is an NL computation. (Note: here “knight” and “rider” are synonyms. German in fact uses the same word *Ritter* for both, whereas the knight in chess is called *Springer* for “jumper.”
And the German word Knecht means "servant" in the sense of "farmhand," while the German word Bauer means "farmer" but literally means "builder" and signifies the pawn in chess.)

(b) Let any instance \((G, s, t)\) of the GAP problem be given, where \(G\) is a directed graph. We create an undirected graph \(G'\) as follows: For every vertex \(u\) of \(G\), \(G'\) has two vertices \(u_1\) and \(u_2\), plus for every directed edge \(e = (u, v)\) of \(G\), \(G'\) has two additional nodes \(u_e\) and \(v_e\). In place of each edge \(e\), \(G'\) has the above-pictured subgraph with \(u_1, u_2\) as at left, then \(u_e\) with capacity 2 and \(v_e\) with capacity 1 in the middle, and finally \(v_1, v_2\) as at right. We do this for all nodes and edges including \(s\) and \(t\).

Suppose \(G\) has a path \(P\) from \(s\) to \(t\). Let's for the moment suppose we start with two riders on the nodes \(s_1\) and \(s_2\) in \(G'\) that correspond to \(s\) (we'll ignore the rule of both starting on one node for now). Suppose the first edge \(e\) in path \(P\) goes to a node \(u\). Then in \(G'\) we have a gadget like the above diagram with nodes \(s_1, s_2, s_e, u_1, u_2\) reading left-to-right. The riders first conquer node \(s_e\) and let's suppose the rider from \(s_1\) at top goes there. That rider can then conquer \(u_e\) all by emself since its capacity is 1. Then the rider still on \(s_2\) can conquer \(u_2\) with help from \(u_e\). This in turn enables the rider on \(u_e\) to conquer the last node \(u_1\) at upper right. Now the riders are set up in the positions they need to begin conquering another gadget in \(G'\) corresponding to the next directed edge \((u, v)\) in \(G\) along the path \(P\). By continuing this process they can conquer all the way to \(t_1\) and \(t_2\), in particular, to \(t_1\).

Conversely, suppose the riders can conquer node \(t_1\) in particular. To escape nodes \(s_1\) and \(s_2\) at all, they can only start by moving to some node \(s_e\) for some edge \(e = (s, u)\) going out of node \(s\). This locks them into the gadget going toward the nodes \(u_1, u_2\) which correspond to \(u\). As observed in the problem statement, they cannot go back to \(s_1, s_2\) through the gadget, so they can only progress by entering another gadget, which corresponds to a directed edge out of \(u\) in \(G\). Thus any actions by the riders in \(G'\) must correspond to tracing out a path in \(G\), so they can conquer \(t_1\) only if \(G\) has a path from \(s\) to \(t\).

The last niggling detail is that in the formal definition of \(L_2\), the riders start on one node—call it \(s_0\)—rather than two nodes \(s_1, s_2\). Putting both riders on \(s_1\) doesn't enable them to conquer \(s_2\) or vice-versa under a strict reading that the conquering must come from \(k = 2\) different adjacent nodes not riders. But what we can do is add an extra root node \(s_0\) and a node \(s'_0\) of capacity only 1, so that one knight on \(s_0\) can move to \(s'_0\) and then connect \(s_0, s'_0\) to \(s_1, s_2\) so that the knights can conquer both of the latter. This sets up the status we started with.

(c) (i) It seems obvious that \(L_\infty\) should be in \(\text{NP}\)—we just guess a sequence of rider movements and verify that in every step where a rider enters a new town, there were enough knights on neighboring towns to conquer it. The rub, however, is the length of the sequence. If they never have to re-conquer a town, then the sequence has at most \(n - 1\) conquests. Since the wording of when they can conquer doesn't depend on how many knights are in a neighboring town, there is no advantage to shuffling them among occupied towns for a particular conquest—but if the army has more than one cluster,
one may need to shift knights between clusters in order to conquer some small-capacity towns before a larger conquest. Still, such shifts require specifying at most \( n \) “from \( x \) to \( y \)” moves between conquests, so the sequence has length at most \( n^2 \) no matter how many knights. That would still put it in \( \text{NP} \). But if towns need to be re-conquered, that argument isn’t enough. An analogy for how the sequence could have exponential length is the more-familiar “Towers of Hanoi” puzzle: This has \( n \) rings of sizes \( 1, \ldots, n \), initially on one of 3 posts in that order top to bottom. A ring can move to another post if the post is empty or a larger ring is uppermost at \( x \). The time to move the bottom ring is order-of \( 2^n \).

Indeed, the riders problem is known to be complete for \( \text{PSPACE} \), hence is deemed unlikely to be in \( \text{NP} \). This was a hard problem for awhile in the 1970s and I can give only a partial explanation. Picture a Turing tape of size \( s = n^{O(1)} \) in which each cell holds 0 or 1. Allocate \( s \) “rungs” as if we were doing an \( \text{NP} \)-completeness proof. The current contents of the tape corresponding to a 0-1 assignment to each cell like before. Now replace each rung by a gadget like the one shown for part (b) but with each node of capacity 2 so that the knights must stay either at the left (value 0) or at the right (value 1). Finally, picture an extra node with connections so that a third knight on it could help the other two conquer the middle and thus switch sides. By moving that knight around we can update the simulated memory at will. The full details of how the logic of those updates is governed are far more complicated, but the upshot is that a polynomial-space bounded Turing machine \( M \) can be simulated by a “logic array graph” of the riders problem.

(ii) For membership in \( \text{DLBA} \) we need not just polynomial space but linear space. The first point is that the size \( N \) of the input graph is really \( m \log n \) if the graph is given by a list of its \( m \) edges, or \( n^2 \) if giving the \( n \times n \) adjacency matrix is less. Either way, presuming \( m \geq n - 1 \) so the graph is connected, a configuration of \( O(n) \) knights needs only \( O(n \log n) = O(N) \) space to specify. So we need only linear space to maintain the current configuration of the knights. Even if the sequence has exponential length each step needs only that much space—and “Towers of Hanoi” has the same situation.

Oh, wait—that’s only enough to show membership in \( \text{NLBA} \). To get \( \text{DLBA} \) we have to search all possible sequences. This is like—but a little trickier than—the membership of the TQBF language in \( \text{DLBA} \). What’s trickier is that whereas each quantified Boolean variable can have at most 2 values, each knight can have up to \( n \) choices of next destination, and there are up to \( n \) knights. This uses \( 2 \log n \) bits to specify each move, but for each branch of a search of depth \( n \) like with \( n \) variables in a QBF, that’s still \( 2n \log(n) = O(N) \) bits to maintain the current branch in a depth-first search. . . Or so I thought when I wrote up the problem, but I forgot here that unlike with TQBF, the branches themselves can have the exponential length from (i). The argument used to show that TQBF was \( \text{PSPACE} \)-complete handles this length by recursively dividing into halves, but it blows up the size to quadratic, so we get no better than \( \text{DSPACE}[n^2] \), which \( \text{Savitch’s Theorem} \) already gives us from \( \text{NLBA} \). Indeed, the intuitive argument for \( \text{PSPACE} \)-hardness given in (i) seems to apply directly when the machine \( M \) is an \( \text{NLBA} \), so the problem would not belong to \( \text{DLBA} \) unless \( \text{NLBA} = \text{DLBA} \), which is open but commonly disbelieved.
Answers to the quantum computing exercises

(2) Suppose $A$ and $B$ are self-adjoint, meaning $A^* = A$ and $B^* = B$. Show that $A \otimes B$ is self-adjoint. Use the concatenation-based indexing convention that $A[u, v]B[w, x] = (A \otimes B)[uw, vx]$. (9 pts.)

\textbf{Answer:} For all indices $u, v$ of $A$ and $w, x$ of $B$,

\[(A \otimes B)^*[uw, vx] = (A \otimes B)[vx, uw]\] (by definition of $^*$)
\[= A[v, u] \cdot B[x, w] \quad \text{(by definition of $\otimes$)}
\[= A^*[u, v] \cdot B^*[w, x] \quad \text{(conjugation is multiplicative)}
\[= A^*[u, v] \cdot B[w, x] \quad \text{(by hyp. $A, B$ self-adjoint)}
\[= (A \otimes B)[uw, vx] \quad \text{(as needed to be proved).}\]

(3) Find a $2 \times 2$ unitary matrix $A$ such that $A^2 = iY$. (The scalar multiple $i$ doesn’t matter in quantum computing and allows $A$ to have real entries. It also doesn’t matter which form of $Y$ is used—either way, $A$ is called the “square root of $Y$.”) 6 pts.

\textbf{Answer:} Take $A = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}$, which is like Hadamard rotated 90 degrees. Then $A^2 = \frac{1}{2} \begin{bmatrix} 0 & 2 \\ -2 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} = iY$. If instead you use $Y = \begin{bmatrix} 0 & i \\ -i & 0 \end{bmatrix}$, then you get $A = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}$ instead. Either way, the operator $A$ is commonly written $Y^{1/2}$ and called the “square root of $Y$.” It and $X^{1/2}$ are the two most common nondeterministic gates (in the standard basis) after Hadamard gates. For contrast, $Z^{1/2} = S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$ (or its conjugate $S^*$ with $-i$ in place of $i$) and is deterministic as well as diagonal (in the standard basis).

(4) Text, exercise 4.13 on page 39. As always expected, show the calculation to justify your answer. (6 pts.)

\textbf{Answer:} We have $F(x, y) = (x, y \oplus f(x))$ for each of the four two-bit combinations. The problem asks to find a one-bit Boolean function $f$ such that $F$ becomes the CNOT function. There are four possibilities for $f$, of which the identity function $f(x) = x$ gives:

\[
\begin{align*}
F(0, 0) &= (0, 0 \oplus f(0)) = (0, 0 \oplus 0) = (0, 0) \\
F(0, 1) &= (0, 1 \oplus f(0)) = (0, 1 \oplus 0) = (0, 1) \\
F(1, 0) &= (1, 0 \oplus f(1)) = (1, 0 \oplus 1) = (1, 1) \\
F(1, 1) &= (1, 1 \oplus f(1)) = (1, 1 \oplus 1) = (1, 0)
\end{align*}
\]

This is the CNOT function.
(5) Consider two-qubit circuits in which line 1 has the H-T-H sequence discussed in lecture in connection with Bell’s Theorem. There are four places to insert a CNOT gate whose control is on line 1. Do all four preserve the property that on input 00, the probability of measuring 0 on line 1 is irrational? (18 pts. total)

**Answer:** If we insert the CNOT gate in front, the qubits on input 00 are unaffected and we still get the irrational probabilities. But if we put it in the second position, we get the following computation on the input vector $A_0 = |00\rangle = (1,0,0,0)^T$, where $\omega = \sqrt{i}$:

\[
A_1 = (H \otimes I)A_0 = \frac{1}{\sqrt{2}}(1,0,1,0)^T
\]

\[
A_2 = \text{CNOT}A_1 = \frac{1}{\sqrt{2}}(1,0,0,1)^T
\]

\[
A_3 = (T \otimes I)A_2 = \frac{1}{\sqrt{2}}(1,0,0,\omega)^T
\]

\[
A_4 = (H \otimes I)A_3 = \frac{1}{2} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \\ \omega \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 \\ \omega \\ 1 \\ -\omega \end{bmatrix}.
\]

What happened with the insertion of the CNOT gate is that it switched the third and fourth entries of $A_1$ so that later when building $A_4$ from $A_3$ they fell into empty slots rather than add together with the first and second entries. Although the final state $A_4$ looks weird with $\omega$, its four entries are all unit complex numbers (multiplied by the outside factor $\frac{1}{2}$), so they all give the same probability—which is just the rational number $\frac{1}{4}$. This is surprising insofar as CNOT is said to be the identity on the first qubit line—but that’s only true when the input is a basis state. When it is a superposition, trickier things can happen—and do happen.

The third position is a mirror image of the second position. Note that $(ABCD)^* = D^*C^*B^*A^*$. Since these matrices are self-adjoint except for $\omega$ becoming $\bar{\omega}$ in the definition of $T$, the reversed computation right-to-left with CNOT in the third position is essentially the same as left-to-right with CNOT in the second position. Again we get the rational probabilities $\frac{1}{4}$. But in the fourth position it is the same as in the first position. (It was also fine to just check this using Davy Wybiral’s quantum circuit simulator or by hand.)

(6) A graph state on $n$ qubits is formed by starting with one Hadamard gate on each line (i.e., starting with the $n$-qubit Hadamard-Walsh transform) and then placing CZ gates as edges of an $n$-vertex graph $G$. Since the CZ gate matrices commute with each other (being diagonal matrices), the order of placing the edges does not matter. The input is $x = 0^n$ and the resulting state can be written as $|G\rangle$.

Note that aside from the normalizing constant $1/2^{n/2}$, every entry in the whole $2^n \times 2^n$ product matrix is 0, +1, or −1. The amplitude of every possible output $z \in \{0,1\}^n$ is hence one of those values. First, state and justify a condition for which value it is. It may help to
think of $z$ as a two-coloring of the vertices of $G$, with 0 for white and 1 for black. Call that amplitude $G[z]$.

For a graph with no nodes, all entries are $+1$ (that is, $+1/2^n$)—indeed, the state is $|+\rangle^n$. For a graph with two nodes and one edge, you can check that three colorings give $+1$ and the both-black coloring gives $-1$; note that this follows the order of entries in the $4 \times 4$ matrix $\text{CZ}$ itself. For a triangle graph with three nodes and three edges, verify that four of the eight colorings give $+1$ and four give $-1$. For the graph with three nodes and two edges—ah, that’s the simplest case of the second part of this problem. . .

This second part applies a second column of Hadamard gates to make a circuit $C_G$. This makes the amplitude of the outcome $0^n$ come from a weighted sum of $G[z]$ over all $z$. In symbols:

$$\langle 0^n | C_G | 0^n \rangle = \frac{1}{2^n} \sum_{z \in \{0,1\}^n} G[z].$$

For the triangle graph, this sum is zero—which means that $C_G$ on input 000 can never give the output 000.

Now finally for the second part of the problem: Suppose we take any graph $G$ and make a graph $G'$ with $n + 2$ nodes by adding a new node $t$ connected only to one vertex $u$ of $G$, and then add another node $s$ connected only to $t$. Show that

$$\langle 0^{n+2} | C_{G'} | 0^{n+2} \rangle = \frac{1}{2} \langle 0^n | C_G | 0^n \rangle.$$

This means that the probability goes down by a factor of 4 but the sign does not change. (6 + 12 = 21 pts., for 99 total on the set)

**Answer:** Since $\text{CZ}$ multiplies by $-1$ only when both qubits carry 1, and we said 1 is the value for coloring a qubit black, we get $-1$ from each edge that has two black nodes in the coloring. Thus overall we get a product of $-1$ not $+1$ whenever the outcome $z$ is a coloring in which there are an odd number of black-black edges. That’s the answer to the first question.

A two-node graph with one edge thus gets $-1$ from the black-black coloring, i.e. $z = 11$, and $(-1)^0 = +1$ from the other three colorings. In the triangle graph, the all-black coloring gives 3 BB edges and the three colorings with one white vertex each give 1 BB edge. So we get 4 cases of $-1$ and 4 cases of $+1$. When we next add the second bank of Hadamard gates to make $C_\Delta$ and do $\langle 000 | C_\Delta | 000 \rangle$ to sum up the amplitudes over all the colorings, everything magically cancels. Thus if you input 000 to the triangle graph-state operator, you can never get 000 as a measured basis output.

To answer the problem’s second question, we can reason about the colorings. Adding the vertices $t$ and $s$ multiplied the number of colorings to consider by four. Suppose that the node $u$ is colored white. The only coloring of $s, t$ that changes the black-edge parity is BB. This means that $3/4$ of the colorings when $u$ is white reinforce the value from $G$ and $1/4$ offset it, for a net multiplication of the numerator of the amplitude by 2. But the denominator got multiplied by 4. If $u$ is colored black, then both colorings with $t$ white add no new black edges. And now the BB coloring adds two new black edges so it also reinforces the value
from $G$. Only the coloring $t = B, s = W$ flips the parity. So in all cases we get that $3/4$ of the colorings in $G'$ augment while $1/4$ offset. So the amplitude of $\langle 0^n \mid C_G \mid 0^n \rangle$ gets multiplied by $\frac{3}{4} = \frac{1}{2}$ when going to $\langle 0^{n+2} \mid C_{G'} \mid 0^{n+2} \rangle$. This finally means that the probability always goes down by $\left(\frac{1}{2}\right)^2 = \frac{1}{4}$, as was to be shown.

For a footnote, consider what happens if $s$ and $t$ are each attached directly to $u$—but not to each other. If $u$ is colored white, then no new black edges are added, so everything reinforces. But if $u$ is black, then there is one way to add nothing, one way to add two black edges which also reinforce, and two ways to add one black edge. The result is a giant humungous cancellation of all cases where $u$ is black—that is, where the outcome in the original graph state (before adding the second bank of Hadamard gates) makes the qubit corresponding to $u$ have the value 1. So what we get is only a reinforcement of the behavior conditioned on $u$ having the value 0 post-measurement. This is like a weird conditioning on the future. Physical experiments verifying such behavior give rise to puzzling philosophical interpretations. Finally, this problem gives a little flavor of how quantum calculations can be aided by shorthands involving graphs and colorings, and that in general one might have to consider contributions from other ways of extending $G$ to $G'$ besides these two. This is ultimately the idea behind Feynman diagrams as calculational aids in physics.