Stefanie Shidoosh
UID: 804794484

CS 180: HW #1

1. The bit complexity of BinaryOneToN is defined by the amount of bit operations with respect to the length of the input. Considering this, in class we determined that the representation of n would be as follows:

$$n = b^{log_b(n)}$$

which is to say that the size of n is $\log(n)$.

Therefore, the number of bit operations this algorithm would execute with respect to the size of n would be $O(log(n))$. It is worth noting that this is intuitive as problems that are solvable in log(n) time often exhibit the property of only one element is needed to execute the algorithm, which was determined in class.

Stefanie Shidoosh
UID: 804794484

2a. A favorable table is one in which at least one column that has an odd sum, as noted. Thus, the algorithm to find the existing unfavorable table from any favorable table goes as follows:

(1) Select the column with the odd amount of 1's

(2) Select a row arbitrarily that has a 1 that is also within the odd amount column, change the 1 to a 0.

(3) If needed, adjust other odd count columns in this manner to produce an even parity.

This will always produce an unfavorable table such that it will produce a table with every column maintaining an even amount of 1's when given a favorable table.

b. Given an input of a favorable table, you can determine whether there exist multiple ways to make the table unfavorable:

Run the algorithm in 2a

If successful, run 2a (2) again but select a different row if such a row exists

If not, there is only one solution

If so, there is at least more than one way to make the table unfavorable.

In this interpretation, one need only to determine if there is at least more than one way. Thus, to find 2 solutions would satisfy finding multiple ways of making the table unfavorable.

c. If given a favorable table, victory is guaranteed with the following algorithm:

If table is favorable and only one row is available, take all matches of that row.

Else, take a match from a row that produces an even parity of 1's column (as detailed in 2a).

When the opponent makes their subsequent move, it will produce a favorable table again, and thus the algorithm repeats until the last match is taken by the opponent.

The algorithms have a time complexity of $O(log(n))$ since only one operation is required once a column with an odd count of 1's is found.

Stefanie Shidoosh
UID: 804794484

3a. Given a graph G = {V, E} that has 2 cycles C_1 and C_2 that each visit half of the vertices exactly once and does not share any vertex, there exists a single cycle in G that visits every vertex in G exactly once.

By definition of a cycle, C_1 and C_2 are both connected graphs, $E(C_1) \subseteq E(G)$ and $E(C_2) \subseteq E(G)$;
$V(C_1) \subseteq V(G)$ and $V(C_2) \subseteq V(G)$ such that $V(C_1)$ contains half of the vertices in the set $V(G)$ and $V(C_2)$ contains the other half, the remaining vertices. $V(C_1) + V(C_2) = \mathrm{V(G)}$.
It is given that there exists a circle $S = \{(v_a, v_b), (v_b, v_c), (v_c, v_d), (v_d, v_a)\}$.
Considering S, it is given that there are more than 2 vertices.
(v_a, v_b) is a part of $C_1$and (v_c, v_d) is a part of C_2, and the remaining edges are not part of either cycle.
The remaining edges, $(v_b, v_c)\ and\ (v_d, v_a)$ connect C_1 and C_2, thus combining these two cycles provides a cycle to the set of all vertices in G, V(G).
Ex: Start the cycle C' at v_a. There exists a cycle, C_1, such that if it starts at v_a it will visit v_b before returning to v_a. There exists the edge (v_b, v_c), so instead of completing the cycle C_1 by returning to v_a, go to v_c. There exists a cycle, C_2, starting at v_c that will visit v_d. Instead of returning to v_c, go to v_a as there exists the edge (v_d, v_a), thus completing the cycle C'.
Therefore, there exists a single cycle in G that visits every vertex in G exactly once.

b. The following algorithm generates a closed knight's tour on a 16 × 16 chessboard, using a closed knight's tour for 8 × 8 as a starting point:

Separate the chess board into 4 quadrants of size 8 x 8
Plot a closed knight's tour in each quadrant
Delete an edge from each tour
Connect (add an edge to) one of the vertices in one closed knight's tour that previously formed the respective removed edge to one of the vertices in the adjacent tour that previously formed its respective removed edge.
Repeat until all tours are connected.
The time complexity is $O(1)$ as the 8x8 tours are provided and the subsequent steps are constant. n is clearly defined to be 16 and thus the time complexity is constant.

c. An algorithm that generates a closed knight's tour on any $2^k\ x\ 2^k$ chessboard for all $k \geq 3$:

Pick an arbitrary cell
Make a possible move for the knight to make from this cell
Recursively check if this move leads to a solution [1]
If it does not, remove this move and try a different one, thus repeating this step. [2]
If all cells are visited, the closed knight's tour is complete.

The time complexity is $O(n^2)$ as it has to follow [1] n times and [2] n times where $n = 2^k$.

Stefanie Shidoosh
UID: 804794484

4. Given an array $B[1..n]$ that stores a binary representation of a positive integer, where each $B[i]$ is either 0 or 1…

Note: 0^{th} element of B as the MSB, n^{th} element as the LSB.

a. A recursive algorithm that evaluates the actual value of the array *B* represents:

Start with first element in array B
If the first element is 1, sum + 1
sum *= 2
Recursively run this procedure with the array $B[i+1 \ ... \ n]$,where i is the current index, and the new sum.

b. An iterative algorithm that evaluates the actual value of the array *B* represents:

For each i^{th} element in B
    If the element is 1, sum + 1
    sum *= 2
    Iterate to $i+1^{th}$ element.

Both algorithms have a time complexity of $O(n)$ as they both require to visit every element in the array of size n.