CSE 4502/5717 Big Data Analytics
Spring 2026; Homework 3 Solutions

1. (a) Note that for a pair of items to be frequent, it has to occur in at least one transaction. Thus the only two-itemsets we have to generate as candidates are the pairs of items we can generate from each transaction. As a result, the number of candidates is $O(n)$. For each such candidate we have to compute the support.

   Support for the candidates can be computed as described in class. In particular, we use a hash tree to store all the candidates. The expected size of each leaf in the hash tree is $O(1)$. Then, we do the following:

   for each transaction T in DB **do**
   for each pair (i, i') of items in T do
   Use the hash tree to increment of the support of (i, i') by 1.
   Output all the pairs of items that have a support of $\geq minSupport$.

   It is clear that the above algorithm has an expected run time of $O(n)$.

   (b) Note that if X is an itemset with k items, then we can form $2^k - 2$ association rules using X. This in turn means that the total number of association rules we can form from a set I of d items is $\sum_{k=2}^{d} \binom{d}{k}(2^k - 2) = \sum_{k=2}^{d} \binom{d}{k}2^k - 2\sum_{k=2}^{d} \binom{d}{k}$ $= 3^d - 2d - 1 - 2(2^d - d - 1) = 3^d - 2^{d+1} + 1$.

2. Let the transactions in the database be $t_1, t_2, \ldots, t_q$. Note that any item in the database can be represented as an integer in the range $[1, n^c]$.

   S is an empty sequence to begin with;
   for $i = 1$ **to** q **do**
   for every item $a \in t_i$ **do**
   Add a to the sequence S;
   Sort S using the integer sorting algorithm;
   Scan through the sorted sequence to count the support for each item and output those that have enough support.

   Clearly, the above algorithm runs in $O(n)$ time.

3. Note that for a pair of items to be frequent, it has to occur in at least one transaction. Thus the only two itemsets we have to generate as candidates are the pairs of items we can generate from each transaction. As a result, the number of candidates is $O(n)$. For each such candidate we have to compute the support.

Let the candidates be $(i_1^1, i_2^1), (i_1^2, i_2^2), \ldots, (i_1^q, i_2^q)$, where $q = O(n)$. We keep any such 2-itemset in sorted order (i.e., $i_1^k < i_2^k$, for $1 \leq k \leq q$). We can think of each such candidate as an integer in the range $[1, n^{2c}]$. We sort these 2-itemsets in lexicographic order using the integer sorting algorithm. This will take $O(n)$ time. We scan through the sorted sequence, identify all the frequent 2-itemsets, and output them.

4. Let $X = x_1, x_2, \ldots, x_n$ be the input. Sort X to get $Y = y_1, y_2, \ldots, y_n$. This takes $O(n \log n)$ time. We can scan through Y and identify all the clusters as follows: Start with x_1. This will be in the first cluster. If the distance between x_1 and x_2 is $\leq \tau$ add x_2 to the first cluster. Keep on adding elements to the first cluster until the distance between the current element x_i and the next one (x_{i+1}) is more than τ. The elements x_1 through x_i will be in the first cluster. x_{i+1} will start the next cluster. Continue in this fashion until all the clusters are identified. The total run time is $O(n \log n)$.

5. The initial state of the register is $|0101\rangle$.

   (a) After the first Hadamard gate, we get: $\frac{1}{\sqrt{2}}(|0101\rangle + |1101\rangle)$.

   (b) The CNOT gate changes the state to: $\frac{1}{\sqrt{2}}(|0101\rangle + |1001\rangle)$.

   (c) After the Hadamard gate we get: $\frac{1}{2}(|0101\rangle + |0111\rangle + |1001\rangle + |1011\rangle$.

   (d) The effect of the next CNOT is: $\frac{1}{2}(|0101\rangle + |0110\rangle + |1001\rangle + |1010\rangle$.

   (e) The next CCNOT has no effect.

   (f) When we apply the two Hadamard gates, the state changes to: $\frac{1}{4}(|0000\rangle - |0001\rangle + |0010\rangle + |0011\rangle - |0100\rangle + |0101\rangle - |0110\rangle + |0111\rangle + |1000\rangle - |1001\rangle + |1010\rangle + |1011\rangle + (|1100\rangle - |1101\rangle + |1110\rangle + |1111\rangle)$.

   (g) After the next CNOT we get: $\frac{1}{4}(|0000\rangle - |0001\rangle + |0010\rangle + |0011\rangle - |0100\rangle + |0101\rangle - |0110\rangle + |0111\rangle + |1000\rangle - |1001\rangle + |1010\rangle + |1011\rangle + (|1100\rangle + |1101\rangle + |1110\rangle - |1111\rangle)$.

   (h) After the final CCNOT, we end up with: $\frac{1}{4}(|0000\rangle - |0001\rangle + |0010\rangle + |0011\rangle - |0100\rangle + |0101\rangle - |0110\rangle + |0111\rangle + |1000\rangle - |1001\rangle + |1010\rangle - |1011\rangle + (|1100\rangle + |1101\rangle + |1110\rangle + |1111\rangle)$.

6. Note that Grover's algorithm solves the following problem: Given a sequence $X = x_1, x_2, \ldots, x_N$ and a function $f : X \to \{0, 1\}$, find an i such that $f(x_i) = 1$. If there are k values of i such that $f(x_i) = 1$, Grover's algorithm solves this problem in $O(\sqrt{N/k})$ time. The output of this circuit will be correct with a constant probability.

   For the satisfiability problem, the input is a Boolean formula F on n variables. The problem is to check if there is a satisfying assignment for F. There are 2^n possible

assignments for the n variables. We can think of X as a set of all possible assignments and $f(.)$ as the function F. For a given assignment, an oracle for $f(.)$ will take $O(|F|)$ time. If there are k satisfying assignments, then, Grover's algorithm will take $O(\sqrt{2^n/k})$ iterations and in each iteration an oracle call will take $O(|F|)$ time and hence, in the worst case (e.g., when $k = 1$), the algorithm will take $O(\sqrt{2^n}|F|)$ time. With a constant probability, given this amount of time, the algorithm will find an assignment (if there is one). We can improve this probability to a high probability by repeating this process $O(\log n)$ times. As a result, the runtime of the algorithm is $\widetilde{O}(\sqrt{2^n}|F| \log n)$.