Solutions to CSE 331 Sample Mid-term

Please do not read anything into the kind of problems in the sample mid-term. In particular, to save myself time I copied two problems from the homeworks. (Further, instead of spelling out the solutions in this document I might refer you to the corresponding homework solutions.) The actual mid term problems will **not** be such straight-forward lifts. Overall, the mid-term will be a bit harder than this sample mid-term (but still much easier than the homeworks). The main purpose of this sample mid-term was to give you an idea of the format of questions. The actual mid term will also have true/false questions followed by two other questions (with same points).

1. ($4 \times 10 = 40$ points) Answer True or False to the following questions and briefly JUSTIFY each answer. A correct answer with no or totally incorrect justification will get you 4 out of the total 10 points. (Recall that a statement is true only if it is logically true in all cases while it is is false if it is not true in some case).

   (a) For any instance of the stable marriage problem with n men and n women, there are $n! = n \times (n-1) \times \ldots \times 1$ many possible perfect matchings.

   True. See Solution to Problem 3 in HW0.

   (b) Consider $f(n) = \log\log n$ and $g(n) = 10^{10^{10^{10^{10^{10}}}}}$. Then $f(n)$ is $O(g(n))$.

   False. $g(n)$ is a constant (albeit a big one) and does not increase with n. However, $f(n)$ does increase with n (though slowly). Thus, for some large enough n, $g(n) \le f(n)$ and thus, $g(n)$ is $O(f(n))$ (and not the other way round).

   (c) For any graph, there is a unique BFS tree for it.

   False. Consider the cycle on four vertices: v_1, v_2, v_3, v_4 such that (v_i, v_{i+1}) for $1 \le i \le 3$ and (v_4, v_1) are the edges. Consider a BFS run that starts from v_1. Note that v_4 can be discovered from either v_2 or v_3 and each choice leads to a different BFS tree.

   (d) Given a graph on n vertices in its adjacency matrix, there is an $O(n^2)$ time algorithm to convert it into its adjacency list representations.

   True. In short here is the algorithm: go through the matrix row by row and for the vertex u corresponding to the current row, add a list of vertices w such that the entry for (u, w) is a 1. Each row takes $O(n)$ time and there are n rows, which makes for a total running time of $O(n^2)$.

2. ($20 + 20 = 40$ points) Given an array A of n integers, consider the the following algorithm that computes a related value (and an intermediate matrix B):

   For every $i = 1, \ldots, n$

       For every $j = i, \ldots, n$

           Assign $B[i, j]$ to be the maximum value among $A[i], A[i+1], \ldots, A[j]$.

   Output the minimum value among all values in $B[i, j]$ (over all $i = 1, \ldots, n$ and $j = i, \ldots, n$).

(a) Prove that the algorithm runs in $O(n^3)$ time.

We are given an algorithm with two nested loops and some operations to perform for each iteration. The outside (i) loop will perform n iterations, which is $O(n)$. For each of those iterations, the inside (j) loop will perform $n - i$ iterations, which is also $O(n)$. So regardless of what is going on inside the inside loop, this algorithm performs $O(n^2)$ iterations. Visually, this should make sense, because we are filling in the upper triangle of the B matrix with values that we compute, and there are $O(n^2)$ entries in the upper triangle. For each of these iterations, though, we find the maximum of $j - i + 1$ (or $O(n)$) terms from the A array. So we have $O(n^2)$ iterations, each of which require $O(n)$ comparisons (which can execute in $O(1)$ time). Thus, the entire algorithm will execute in $O(n^3)$ time.

(b) Present another algorithm that solves the same problem but runs in $O(n^2)$ time. (Briefly justify the running time and correctness of your algorithm.)

We can develop a more efficient algorithm by isolating and removing any unnecessary or duplicated operations from the given algorithm. To see where these exist, consider that for each iteration, we go back to the A array and found out the maximum of a bunch of entries, ignoring the fact that in the previous iterations, we've found the maximum of many of those same terms together already. Notice that for any element of B except when $j = i + 1$, the entry in $B[i, j]$ equals $\max(B[i, j-1], A[j])$. This suggests the following algorithm:

---

for $i = 1, 2, ..., n-1$ **do**
  for $j = i+1, i+2, ..., n$ **do**
    if $j = i + 1$ **then**
      $B[i, j] = \max(A[i], A[j])$
    else
      $B[i, j] = \max(B[i, j-1], A[j])$
    end if
  end for
end for

Although the structure of the algorithm is similar to the previous one, we have only one comparison per iteration, instead of the earlier $O(n)$ comparisons. So the time complexity function for this algorithm just depends on the number of iterations, which is $O(n^2)$. The running time is thus $O(n^2)$.

3. Let $d \geq 1$ be an integer. Then a d-dimension *hypercube* is a graph whose vertex set is $\{0, 1\}^d$. (Note that this implies that $n = 2^d$.) Further, a pair (u, v) is an edge if and only if the binary representations of u and v differ in exactly one of the d positions.

   (a) (20 points) Figure out a function $f(d)$ such that the d-dimension hypercube has a cycle of length at least $f(d)$. (You will get more points the larger the value $f(d)$ is.) Briefly justify your answer.

(*Hint:* You can assume the existence of the *Gray code*, which for any $\ell \geq 1$, outputs an ordering of binary vector of length ℓ such that one can go from one vector to the next one in the ordering by flipping exactly one bit.)

We claim that $f(d) = 2^d$, i.e. there exists a cycle that contains all the vertices in the graph. (Such a cycle is called a *Hamiltonian cycle*.) Let the Gray code ordering of the n vertices be $v_1, \ldots, v_n$. We claim that $v_1, v_2, \ldots, v_n, v_1$ is a cycle. To show this we must argue that (v_i, v_{i+1}) and (v_n, v_1) are all edges. This is true by the definition of the Gray code and the hypergraph. For $1 \leq i \leq n-1$, v_{i+1} can be obtained from v_i by flipping one bit, i.e. v_{i+1} and v_i differ in exactly one of the d positions and hence, (v_i, v_{i+1}) is an edge. A similar argument shows that (v_n, v_1) is an edge.

(b) (**Bonus**) (10 points) A *cut* of a graph $G = (V, E)$ is a partition of V into two sets S and $\bar{S} = V \setminus S$. The *value* of a cut $(S, \bar{S})$ (denoted by $E(S, \bar{S})$) is the total number of edges such that one end point is in S and the other is in $\bar{S}$, i.e. it is the number of edges "crossing" the cut. The *maxcut value* of G is

$$\max_{S \subseteq V} E(S, \bar{S}).$$

Figure out a function $g(d)$ such that the maxcut value of a d-dimension hypergraph is at least $g(d)$. Justify your answer. (To receive *any* credit for this problem, the function $g(d)$ has to depend non-trivially on d– at the very least it has to be asymptotically bigger than d. An answer without any justification will not receive any credit.)

If you thought about this problem, contact us with your solution and we will be happy to talk about it.