Midterm 1

- Do not open this quiz booklet until you are directed to do so. Read all the instructions first.
- This quiz contains 4 problems. You have 120 minutes to complete all problems. The points for each problem in the midterm roughly indicate how many minutes you should spend on the problem.
- This quiz booklet contains 12 pages, including this one and 3 sheets of scratch paper. **Please do not remove any pages from the booklet, including the scratch papers!**
- Write your solutions in the space provided. If you continue your solution on the back of any page, please make a note of it. If you run out of space, continue on a scratch page and make a note that you have done so.
- Do not waste time deriving facts that we have studied. Just cite results from class.
- When we ask you to "give an algorithm" in this quiz, describe your algorithm in plain English or if necessary, pseudocode, and provide a short argument for correctness and running time. You do not need to provide a diagram or example unless it helps make your explanation clearer.
- Do not spend too much time on any one problem.
- **Please write your name on every single page of this exam.**
- Good luck!

Problem	Title	Points	Parts	Grade	Initials
0	Name	1	1		
1	True/False Questions	30	7		
2	Majority Element	30	1		
3	Parenthesizing	30	1		
4	Planting Flowers	30	1		
Total		121			

Quiz 1-1: [30 points] True/False Questions

Mark each statement as either true or false and **provide a short explanation for each answer**.

(a) [5 points] The median of medians algorithm, when run on groups of size 7 instead of 5, has the recurrence $T(n) = T\left(\frac{n}{7}\right) + T\left(\frac{5n}{7}\right) + O(n)$.

Solution: True.There will now be $\frac{n}{7}$ groups of size 7 each. And we know that at least $4 \times \frac{1}{2} \times \frac{n}{7} = \frac{2n}{7}$ elements will be on either side of the median of medians after partitioning. Hence we recurse on a list of size $n - \frac{2n}{7} = \frac{5n}{7}$ elements in the worst case.

(b) [4 points] If the primal Linear Program is feasible and has a bounded optimum, then the dual Linear Program is feasible.

Solution: True. By the LP duality theorem.

(c) [4 points] The majority of the work in the mergesort algorithm is done at the root of the recurrence tree.

Solution: False. The work is evenly distributed across the levels of the recurrence tree. Each level involves $O(n)$ work with a total of $O(\log(n))$.

(d) [4 points] Let $G = (V, E)$ be a connected graph with unique edge weights where $|E| \geq 2$ with at most one edge between each pair of vertices. Then the edge with the second smallest weight must be in the MST for G.

Solution: True. In Kruskal's algorithm, the second smallest weight edge will be chosen for the MST since it does not connect the same pair of vertices as the smallest weight edge.

(e) [4 points] If an edge e is in the MST for a graph G, then it must be a minimum weight edge across some cut in G.

Solution: True. Consider the cut that separates the vertices in the same way as the cut that is formed by removing e from the MST of G. If there were a smaller weight edge across this cut than e, we can simply replace e with this edge and get a better MST.

(f) [4 points] Given two sets of integers X and Y, such that $|X| = |Y| = n$, we can compute their Minkowski Sum $\{x + y | x \in X, y \in Y\}$ using FFT in $O(n \log n)$ time.

Solution: False. The FFT runs in $O(m \log m)$ time where m is a bound on the integers themselves, not on the size of the sets.

(g) [5 points] *In Johnson's algorithm for APSP in a graph G, recall that we compute the re-weighting function $h : V \to \mathbb{R}$ by introducing a new vertex s and adding an edge of weight 0 from s to all the vertices in G. We then run Bellman-Ford from s and set $h(u)$ to be the shortest path from s to u.*

Instead of 0, we can set the weight of the new edges from s to be an arbitrary constant $c \neq 0$ and still obtain a function h that works.

Solution: True. The desired property is that $w(u, v) + h(u) - h(v) \geq 0$. Hence, no matter what the weight of the new edges is, the triangle inequality still holds that $h(v) \leq h(u) + w(u, v)$ for any shortest path h.

Quiz 1-2: [30 points] Majority Element

Suppose we have a list of n objects, where $n = 2^k$ for some positive integer k. The only operation that we can use on these objects is the following: $isEqual(i, j)$, which runs in constant time and returns True if and only if two objects i and j are equal. The objects are neither hashable nor comparable.

Our goal is to find if there exists a majority (more than $n/2$ occurrences) of the same element and if so, return it.

Design an $O(n)$ time algorithm that finds whether a majority element exists, and if so, returns it. Provide arguments for your algorithm's correctness and runtime.

Note: A fully correct $O(n \log n)$ time algorithm with correctness and runtime arguments will receive 25 / 30 points.

Hint for $O(n)$ solution: Consider implementing and using a subroutine $Reduce(L)$ which converts the list L into an $n/2$ sized list of each adjacent pairs.

For an example list L, where

$$L = [\ddagger, \oslash, \star, \ddagger, \sqcap, \sqcap, \ddagger, \ddagger, \ddagger, \oslash, \ddagger, \bigcirc, \ddagger, \uplus, \ddagger, \ddagger]$$

A single iteration of $Reduce(L)$ would create a list L' as follows:

$$L' = [(\ddagger, \oslash), (\star, \ddagger), (\sqcap, \sqcap), (\ddagger, \ddagger), (\ddagger, \oslash), (\ddagger, \bigcirc), (\ddagger, \uplus), (\ddagger, \ddagger)]$$

Solution: We want to use divide and conquer to solve this problem - given our list L, we want to create a list M where if there is a majority element $x \in L$, x will be a majority element in M.

For a linear time algorithm, we can do the following: Run the operation $Reduce(L)$, which reduces our size n list to an $n/2$ list of object pairs L'.

For a given object pair (i, j), we run $isEqual(i, j)$. If $i == j$, then we include i in our new list M. If not, we discard both elements.

Since we discard at least half the elements in L, M has size $\leq n/2$.

We keep recursing until we reach a list of size at most 2, where we can find the majority in constant time. If the list is of size one, then we return the single element. If the list is of size 2, we return the first element iff the two elements in the list are equal. If not, we return None, since there is no majority.

If we find that M (our subproblem) returns a candidate majority element y, in linear time, we scan L to see if it's truly a majority element in L. If so, we return y - if not, we return None.

Since M has size at most $n/2$, our recurrence becomes:

$$T(n) = T(n/2) + O(n)$$

which is O(n) time.

Correctness: To show that our algorithm is correct, we have to show that if there is a majority element $x \in L$, x is a majority element in M. To do this, we look at the subroutine $Reduce$.

$Reduce$ creates object pairs (i, j).

If $i \neq j$, we discard both i and j. Since there are at most $n/4$ occurrences of x in such pairs - at most $n/2$ such pairs can occur in total - we discard at most $n/4$ occurrences of x.

For x to be a majority element, there must be $n/2 + c$ occurrences of it, for some positive c. If we lose $n/4$ of these occurrences, there must still be $n/4 + c$ occurrences of x in M. Note that including one element of each pair, if the pair is made of equal elements does not affect the majority - since we are just dividing all frequencies by 2.

Since M has size at most $n/2$, x will still be a majority element in M.

Quiz 1-3: [30 points] Parenthesizing

Ben is preparing a homework for his students to practice arithmetic operations. He has written down a sequence of n numbers $a_1, a_2, \ldots, a_n$ with $n-1$ arithmetic operations $O_1, O_2, .. O_{n-1}$ between each pair of consecutive numbers, where each O_i is an addition ($+$) or multiplication ($\times$). The expression looks like:

$$a_1\, O_1\, a_2\, O_2\, ..\, O_{n-1}\, a_n$$

He notices that the final answer depends on how one may parenthesize the expression. Unfortunately, Ben suffers from sevenophilia. Thus, he won't be happy with the assignment unless the final answer is zero modulo seven. Alice wants to help him by adding exactly $n-1$ pairs of parentheses (one for each operation) to the expression, such that the final answer becomes zero modulo seven. It turns out there may be many possible ways to do this.

Give an $O(n^3)$ time algorithm that finds how many ways there are to add parentheses to the expression so that the final expression evaluates to zero modulo seven.

Example: Given the expression $5 + 2 \times 3$, the solution is 1. This is because there is only one way to parenthesize the expression to get $0 \mod 7$, namely $((5 + 2) \times 3) \equiv 0 \mod 7$.

Note: You may use without proof that $a \times b \mod p = (a \mod p) \times (b \mod p) \mod p$.

Solution: We use dynamic programming to solve this problem. Let $D[i][j][x]$ be the number of ways to parenthesize the sub-expression from a_i to a_j so that it yields a final answer of x modulo seven.

Base case: $D[i][i][x]$ is 1 if a_i modulo 7 is x. Otherwise, it is 0.

Assume we want to compute the value of $D[i][j][x]$. Each way of parenthesizing this expression yields a unique sequence of evaluating each operation. Consider the last operation in this sequence. It can be the evaluation of any of the $j - i$ operators in the expression. If the k-th operator is the last one, we can consider the value of the expression to be the value of expression $a_i, \ldots, a_{i+k-1}$ plus (or multiply by) the value of the expression $a_{i+k}, \ldots, a_j$.

For example, given the expression $5 + 2 \times 3 + 1$, the operation can be one of the following:

$$5 + 2 \times 3 + 1 \rightarrow \begin{cases} (5 + (2 \times 3 + 1)) \\ ((5 + 2) \times (3 + 1)) \\ ((5 + 2 \times 3) + 1) \end{cases}$$

Thus, we can compute $D[i][j][x]$ by iterating over all possibilities for these operands. Initially, $D[i][j][x]$ is zero. Let y and z be two values in $\{1, 2, \ldots, 6\}$ such that $y + z$ (or $y \times z$

if the k-th operator is $\times$) is equal to x modulo 7. For any such pair y and z, we update the value of $D[i][j][x]$ according to the following:

$$D[i][j][x] \leftarrow D[i][j][x] + D[i][i+k-1][y] \times D[i+k][j][z]$$

Therefore, we our counting all possible ways of constructing expressions of value $x \mod 7$. The final solution to the problem will then be $D[1][n][0]$.

Running time: The dynamic programming table has $O(n^2)$ cells and we spend $O(n)$ times to fill each of these cells. Therefore, the total running time is $O(n^3)$.

Quiz 1-4: [30 points] Planting Flowers

Ben Bitdiddle wants to plant flowers in his garden, which is a grid with n rows and m columns. He learns that city regulations require that each row i can have at most r_i flowers and each column j can have at most c_j flowers. Subject to these rules, Ben wants to plant as many flowers in his garden as possible.

Unfortunately some grid cells in his garden have large rocks in them, so there is no space for flowers. All other grid cells can each contain up to d flowers. You may assume you are given $b_{i,j}$ where $b_{i,j} = 1$ if cell (i, j) has space for flowers, and 0 otherwise (if there is a large rock there).

Given all d, r_i, c_j and $b_{i,j}$, find an $O(n^2m^2d)$ algorithm that outputs how many flowers to plant in each cell such that the number of flowers is maximized. Argue the correctness of your algorithm and its runtime.

Hint: Consider constructing a flow network.

Solution: We build a flow network as follows:

- Add a node for each row i.
- Add a node for each column j.
- Add a source s and a sink t.
- Add an edge with capacity r_i from s to the node for row i.
- Add an edge with capacity c_j from the node for column j to t.
- If cell (i, j) is empty, add an edge with capacity d from the node for row i to the node for column j.

The maximum flow on this graph will give us the maximum number of flowers that can be planted without exceeding constraints. The flow from the node for row i to the node for column j is exactly how many flowers Ben should plant in the cell (i, j). To check if this garden is perfect, we can either check the sums at the end, or we can verify that every edge connected to the source and sink are fully saturated. This works because a fully saturated edge from s to one of the row nodes means that the number of flowers in that row is maximized. The same logic works on the column nodes.

Correctness: The edges coming into each row node and exiting each column node bound the number of flowers that can be planted in each row/column, while the edges between the row and column nodes bound how many flowers can be planted in each cell.

Runtime analysis: the maximum number of edges is nm. The maximum flow possible happens if each grid cell gets d flowers, for a total of nmd. Ford-Fulkerson will give us a runtime of $O(|E|f) = O(n^2m^2d)$. Edmonds-Karp will give $O(VE^2) = O((n + m)n^2m^2)$, which could be better or worse depending on how d compares to $n + m$.

Scratch page

Scratch page

Scratch page