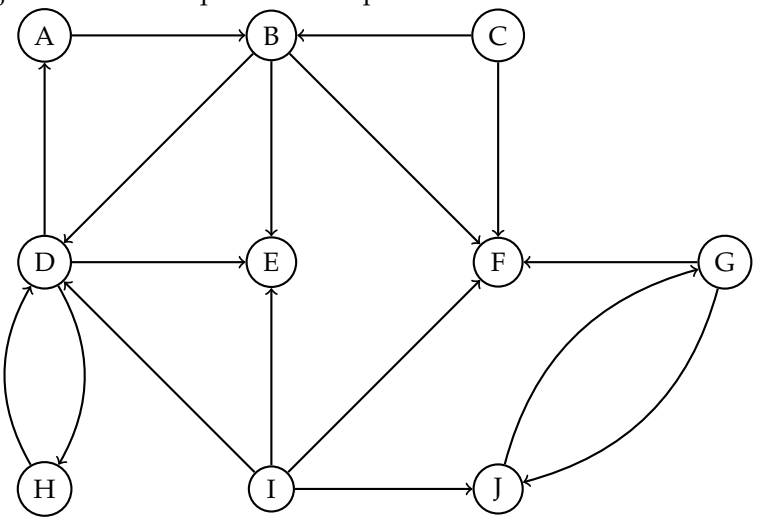
Midterm 1

1 Connectivity in Graphs

No justification is required on this problem.



(a) (**2 points**) For the directed graph above, write down its strongly connected components (SCCs) in any topological order.

Please write the solution in the form (for example) [(ABD), (CE), F, (GH)].

(b) (**3 points**) **True or False:** There exists one edge that would make the graph have exactly two source SCCs and one sink SCC.

Solution:

(a) There are 6 strongly connected components. A topological ordering is.
[C, I, (ABDH), E, (GJ), F]

(b) True. The original graph has two sources, C and I, and two sinks, E and F. There are numerous answers, but all possible answers use exactly one edge beginning at E or F and not ending on C or I. Possible answers include sink to sink (such as (E, F) or (F, E)), sink to any node in (ABDH), or E to any node in (GJ), but not (F, G) or (F, J).

2 True/False

No justification is needed nor will be examined. DFS means depth first search, and BFS means breadth first search. **(3 points each.)**

(a) n^2 is in $O(n^3)$.
Solution: True. $lim_{n\to\infty} \frac{n^2}{n^3} \to 0$.

(b) Let $T(n) = 4T(n/2) + n$ and $S(n) = 3S(n/2) + n$. Then $T(n) = O(S(n))$.
Solution: False. $S(n)$ is $O(n^{\log_2 3})$ $\Theta(n^2)$ describes $T(n)$.

(c) In a DFS on a directed graph $G = (V, E)$, if u was explored before v, and there is a path from u to v, $post(v)$ is greater than $post(u)$.
Solution: False. $post(v)$ is less than $post(u)$ since v will be explored while u is still on the stack.

(d) Suppose DFS is called on a DAG G, and the resulting DFS forest has a single tree. Then G has exactly one source vertex.
Solution: True. Any source can only be explored from the main depth-first search procedure.

(e) If a root r in a DFS tree of undirected graph G has more than one child, then $G - r$ (the graph resulting from removing r and any incident edges from G) has more than one connected component. Solution: True. This is from the homework. Basically, there is no path from the descendants of the first child to any subsequent child that does not use the root r, as explore would have explored that path.

(f) Let G be an undirected graph and G^* be the result of removing some edge (u, v) from G. Suppose that G, G^* are connected, and let T be a BFS tree of G rooted at s, and T^* a BFS tree of G^* rooted at s. If for all vertices w, the depth of w in T is the same as the depth of w in T^*, then u and v are at the same depth in T.
Solution: False. It is possible that (u, v) is a (non-tree) edge between vertices in two adjacent levels.

(g) Let T be any tree which contains all the vertices of a connected undirected graph G. There is a way to break ties in DFS that will output T.
Solution: False. Consider a complete graph, any depth first tree is a path visiting every vertex.

(h) Let T be any tree which contains all the vertices of a connected undirected graph G. There is a way to break ties in BFS that will output T.
Solution: False. The BFS of a complete graph is a single node connected to all the others which cannot be a path.

(i) Suppose G is strongly connected, with integer edge weights, and has a shortest paths from some vertex v (i.e. a finite weight shortest path exists to all nodes from v). Then there are shortest paths from every vertex to every other vertex.
Solution: True. The existence of a shortest path tree in a strongly connected graph ensures there is no negative cycle, thus there must be a shortest path tree from any source.

(j) Suppose a directed graph G has integer edge weights and has well-defined shortest path tree (SPT) from some source s (i.e. a finite shortest path exists to all nodes). Bellman-Ford always returns a correct SPT from any vertex.
Solution:
True. There is no negative cycle.

(k) Suppose G is a DAG. The longest path can be found by negating all edge lengths and then running Dijkstra's algorithm from every source node.
Solution: FALSE. In a DAG, the longest path is well defined and can certainly be found by negating all the edges and updating in topological order. Consider a graph with edges (a, b) of weight 1, an edge

(a, c) of weight 3, an edge from (b, c) of weight 5, and an edge from (c, d) of weight 1. Dijkstra's will label d with -4 corresponding to a path from a to c to d, where the longest path has length corresponds to -7 which is the path a, b, c, d.

3 Short Answer

No need to justify. Anything outside the box will not be graded unless you **clearly** indicate that your answer is elsewhere (e.g., with an arrow from inside the box). You should try to avoid doing this, though.
(4 points each.)

(a) If each of the integers a and b uses n-bits in their binary representations, what is the **maximum** number of bits that are needed to represent $a \times b$? (An exact expression in terms of n should be given.)

Solution: $2n$. The number of bits is within one of $\log ab = \log a + \log b \leq 2n$.

For the following three parts, solve for $T(n)$. Write your answer using Θ notation. Assume $T(c) = \Theta(1)$ for any suitable constant c.

(b) $T(n) = 9T(n/3) + n^2$.

Solution: We can apply the Master theorem. Since $\log_b a = \log_3 9 = 2$, the solution to the recurrence is

$$\boxed{\Theta(n^2 \log n)}$$

(c) $T(n) = 2T(n/2) + n \log n$

Solution: Expanding the recurrence at lower levels and back-substituting yields the following:

$$T(n) = 2^i T(n/2^i) + n \sum_{j=0}^{i-1} \log \frac{n}{2^j}$$

Setting $i := \log n$,

$$T(n) = nT(1) + n \sum_{j=0}^{\log n - 1} \log \frac{n}{2^j}$$

. The second term is at most $n \log^2 n$ by noticing that there are $\log n$ terms all of which are at most $n \log n$. Moreover, there are $\log n/2$ terms that are at least $n \log(\sqrt{n}) = (n \log n)/2$ by looking at the larger half of the summation. Thus, the second summation is $\Theta(n \log^2 n$. Furthermore, $nT(1)$ is $\Theta(n)$, so $T(n)$ is $\Theta(n \log^2 n)$.

$$\boxed{\Theta(n \log^2 n)}$$

(d) $T(n) = 4T(\sqrt[4]{n}) + \log n$.

Solution: Expanding the recurrence at lower levels and back-substituting yields the following:

$$T(n) = 4^i T(n^{1/4^i}) + i \log n$$

The recursion stops at depth $i = \log\log n$ if make the assumption that our base case is $T(4) = c$, yielding a final expression for $T(n)$ as below:

$$T(n) = \log n T(4) + (\log\log n)(\log n)$$

Therefore, $T(n)$ is $\Theta((\log\log n)(\log n))$ because the dominating factor in the final expression is $(\log\log n)(\log n)$ which we can prove by evaluating $\lim_{n\to\infty} \frac{(\log\log n)(\log n)}{\log n}$.

$$\boxed{\Theta((\log\log n)(\log n))}$$

(e) What is the largest number of directed edges in an n-vertex DAG?

Solution: $\binom{n}{2}$. Write them down in a linear order, and add all possible directed edges from "earlier" to "later.

(f) Given an n-vertex DAG with a single source, what is the maximum number of trees in the DFS search forest among any ordering of the vertices (in DFS)?

Solution: n. If the vertices are arranged in reverse topological order each call of explore from depth-first-search only explores one vertex.

(g) Consider an undirected graph with n vertices, and m edges, and maximum degree Δ. Run Dijkstra's algorithm using the array $d[\cdot]$ to maintain distance labels. What is the maximum number of times that $d[v]$ can be decreased for any fixed vertex v?

Solution: Δ. Each neighbor can find a new distance. $\Delta - 1$ is acceptable as one may not view decreasing from ∞ to be a decrease.

4 Quick Fixes: Graphs

Your answers here should be one or two sentences. A clear statement of the answer or algorithm is fine.

(a) Consider a depth first search of a **directed** graph G that gives a $pre[\cdot]$ and $post[\cdot]$ numbering and the set of roots of each tree output.

(i) (**4 points**) Consider deleting a single root r and its incident edges from G to form G'. Describe how to modify the pre and post arrays to give valid ones for G' for depth first search where vertices are ordered with respect to the pre-ordering number. (Note that you have no access to G', you just have access to the arrays.)

Solution: Subtract one from the pre,post numbers for any vertex whose intervals are in $[pre[r], post[r]]$ and subtract 2 from the pre,post for any vertex whose $pre[v] > post[r]$. Leave the others alone.

The notion is that we are simply removing the visit to r from the traversal at the appropriate points in time.

(ii) (**4 points**) Recall that a forward and tree edge $e = (u, v)$ both have $pre[u] < pre[v] < post[v] < post[u]$. Give a method to distinguish whether an edge $e = (u, v)$ is a forward or tree edge using only the pre and post numberings (i.e., without knowledge of any other edge.)

Solution: For a vertex u, if there is vertex z where $pre[u] < pre[z] < pre[v] < post[v] < post[z] < post[u]$, then the edge (u, v) is a forward edge. Otherwise it is a tree edge. That is, (u, z) must be a tree edge and z is an ancestor of v.

(b) You are given a directed, strongly connected graph $G = (V, E)$ with **possibly negative** but finite edge weights, $\ell(u, v)$ for directed edge $e = (u, v)$, and an array d indexed by vertices in V.

(i) (**4 points**) Give a linear ($O(|V| + |E|)$) time algorithm to *check* whether the array d corresponds to shortest path distances from a vertex s. (Your algorithm should use $d(\cdot)$, $\ell(u, v)$ for each edge $e = (u, v)$ and an access to $d(v)$ is $O(1)$ time for any vertex v.)

Solution: Check if $d(u) + \ell(u, v) \geq (d(v))$ for each edge $e = (u, v)$, whether $d(s) = 0$, and that for each $v \neq s$, there is a $e = (u, v)$ where $d(v) = d(u) + l(u, v)$.

The first condition shows that no distance is too large, the second is the base case, and the third ensures that the distances correspond to the length of a shortest path.

(ii) (**4 points**) Now suppose that $d(\cdot)$ is a valid set of shortest path distances from s. Show how to use Dijkstra's algorithm to compute the shortest distances from some other $s' \in V$. Be sure to describe how to recover the actual distances from your computation.

(Hint: define a new weight for each edge $e = (u, v)$ for the graph G using $d(\cdot)$ and the previous weight $\ell(e)$. Also, notice if $x \geq y$ that $x - y \geq 0$.)

Solution: Since, we have $d(u) + \ell(u, v) \geq d(v)$ for each edge $e = (u, v)$, the quantity $\ell'(u, v) = \ell(u, v) - d(v) + d(u) \geq 0$.

Computing distances according to $\ell'(v)$ from s', yields the shortest paths $d'(v)$ from s under $\ell'(\cdot)$.

Now for a path P from s' to v,

$$\sum_{e \in P} \ell'(e) = \sum_{e=(u',v') \in P} (d(u') + \ell(u', v') - d(v')) = d(s') - d(v) + \sum_{e \in P} \ell(e).$$

Notice that the path that minimizes this expression for s' and v does not depend on $d(s)$ and $d(v)$ but only on the length of the path as $d(s') - d(v)$ is included in $\ell'(P)$ for every path. Thus $d'(v)$ corresponds to $d(s) - d(v) +$ the length of the shortest u to v path.

Thus, the distance to v from s' is $d'(v) + d(v) - d(s')$.

5 Fast Fourier Transform

(a) (**6 points**) Suppose that $FT(a_0, a_2) = (1, 1)$ and $FT(a_1, a_3) = (1, 2)$. What is $FT(a_0, a_1, a_2, a_3)$? (Recall $FT(a_0, a_1, a_2, a_3)$ is the evaluation of the polynomial $a_0 + a_1 x + a_2 x^2 + a_3 x^3$ on $\{i^0, i, i^2, i^3\}$.)

Solution: $(2, 1 + 2i, 0, 1 - 2i)$.

This is an $n = 4$ FFT.

Recall $FFT(a_0, a_1, a_2, a_3)[i] = FFT(a_0, a_2)[i] + \omega^i FFT(a_1, a_3)[i]$, for $i < 2$ and $FFT(a_0, a_1, a_2, a_3)[i] = FFT(a_0, a_2)[i - 2] - \omega^i FFT(a_1, a_3)[i - 2]$, for $i \geq 2$, where $\omega = i$ for $n = 4$.

So, we get $(1 + 1, 1 + 2i, 1 - 1, 1 - 2i) = (2, 1 + 2i, 0, 1 - 2i)$.

(b) Let $a_0, \ldots, a_{n-1}$ be some complex numbers.

$$A = (A_0, \ldots, A_{n-1}) = FT(a_0, \ldots, a_{n-1}) \quad \text{and} \quad A' = (A'_0, \ldots, A'_{n-1}) = FT(a'_0, a_1, \ldots, a_{n-1})$$

That is, A' is the FFT of a sequence with exactly one element changed.

(i) (**5 points**) How many values of A and A' differ if $a_0 \neq a'_0$?

Solution: n. As noted in the solution below: $A'(x) = A(x) - a_0 + a'_0$ which means they differ on any point by $a_0 - a'_0$, and in particular on the n roots of unity that FT evaluates them on.

(ii) (**5 points**) Consider that $(A_0, \ldots, A_n) = FT(a_0, \ldots, a_{n-1})$, give a $O(n)$ (linear) time algorithm to compute $(A'_0, \ldots, A'_n) = FT(a'_0, a_1, \ldots, a_{n-1})$.

Solution: Recall that $FT(a_0, \ldots, a_{n-1})$ is the evaluation of $A(x) = a_0 + \cdots a_{n-1}x^{n-1}$ on $1, \omega, \ldots, \omega^{n-1}$. Thus, the difference for $FT(a'_0, a_1, \ldots, a_n)$ is that the polynomial is $A'(x) = a'_0 + a_1 x + \cdots a_{n-1}x^{n-1}$. Thus $A'(x) = A(x) - a_0 + a'_0$. Thus, one can update each $A'(\omega^i)$ with a single addition/subtraction.

6 A cool set encoding

Given a sequence of integers $S = s_1, \ldots, s_k$, there is a curious way to treat it as a polynomial: $S(x) = \sum_{i=1}^{k} x^{s_i}$. Note that if several elements have the same value v, the x^v term will appear several times and can be collected together. If one wishes to see how many elements of the sequence are equal to v, then one can simply look at the coefficent of x^v in $P(x)$.

Given three sets R, S, T, each of size n and containing integers in the range $[0, n]$, we wish to find the number of solutions to the equation $r + s + t = z$ where $r \in R, s \in S$ and $t \in T$.

(a) (**2 points**) Give a straightforward $O(n^3)$ time algorithm to find the number of solutions to this equation. (Your description should be very brief.)

Solution: We can simply check all $O(n^3)$ combinations of $i, j, k \in [0, n-1]$ whether $r_i + s_j + t_k = z$.

(b) (**8 points**) Give a more efficient algorithm for this problem. (Here describe your algorithm, argue correctness, and give a runtime bound.)

Solution: We can compute the polynomials $R(x), S(x)$ and $T(x)$ from the sets as described above. Then multiply them together to produce $P(x) = R(x)S(x)T(x)$. Output the coefficient of x^z.

This follows from thinking about each term when multiplying out.

$(x^{r_0} + \cdots + x^{r_{n-1}})(x^{s_0} + \cdots x^{s_{n-1}})(x^{t_0} + \cdots x^{t_{n-1}})$.

The product can be obtained by adding all the terms of the form $x^{r_i}x^{s_j}x^{t_k}$ for all $i, j, k \in [0, n-1]$. Notice that this is just $x^{r_i+s_j+t_k}$. So the coefficient of x^z is simply the number of terms where $x^{r_i+s_j+t_k} = x^z$ or where $z = r_i + s_j + t_k$.

Since all the integers have values in the range 1 to n, the degree of all the polynomials is at most n and the degree of the product is at most $3n$.

Thus, we can use the FFT with a sequence of coefficients length larger than $3n$ to perform the multiplications in $O(n \log n)$ time.

7 Deterministic Selection

You will design a deterministic algorithm for selecting the kth smallest element of n numbers in some set S. (Assume that the numbers are all distinct. You may describe your method as a modification to an algorithm presented in class or the book.)

(a) (**4 points**) Group the n numbers in S into groups of 7, and take the median of each to form the set M. Now, take the median x of M. Give a lower bound on the number of elements in S that are smaller than x. (You can assume that n is divisible by 7 or not worry about rounding, i.e., you can assume $\lfloor n/7 \rfloor = n/7$)

Solution: x is larger than $4\lfloor n/7 \rfloor (1/2) > 2n/7 - 4$ elements as it is larger than the median of half the sets which says that 4 elements in each set are smaller than it. Thus, it is larger than 4 elements in each of $(1/2)(n/7)$ lsits. Rounding the $n/7$ loses 4 elements at most, and the rounding for $1/2$ actually helps. Assuming it is divisible by 7, we get $2n/7$ as the lower bound.

(b) (**6 points**) Briefly describe a linear time algorithm for computing the kth element of S. If you use a recursive algorithm, please provide a recurrence for the runtime of your algorithm. (Again, for recursive algorithms, don't worry about rounding.)

Solution: Perform the above operation and then use the element x in place of the randomly selected partition element for the Selection algorithm from class. The recurrence is $T(n) = T(n/7) + T(5n/7) + O(n)$ where $T(n/7)$ is the recursive call to find the median of the medians and $T(5n/7)$ is the recursive call on the remaining elements in the Select procedure. $T(n)$ is $O(n)$ since one gets a geometric sum as one unfolds it with constant $(6/7)$.

8 You are not better than me! (Pareto Optimality)

A point (x, y) is dominated by another point (x', y') if $x < x'$ and $y < y'$.

Given a set of points $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, an undominated point $p = (x_i, y_i)$ is one which is not dominated by any other point in S. We wish to find an algorithm for finding all the undominated points. (Assume that for all $i \neq j \in \{1, \ldots, n\}$, $x_i \neq x_j$ and $y_i \neq y_j$, i.e., all x and y values are distinct.)

(a) (**2 points**) Give a straightforward $O(n^2)$ time algorithm for finding the undominated points. (Just the idea here, it should be a sentence.)

Solution: For each point, check whether any other point dominates it. Two enclosing loops over n things, gives $O(n^2)$ time. Frankly, this part is to make sure students understand the question.

(b) (**4 points**) Let m_x be the median value of the x-values of the points and define $S_0 = \{(x, y) \in S : x \geq m_x\}$ and $S_1 = \{(x, y) \in S : x < m_x\}$. That is, S_0 contains all points with x value *at least* the median x-value, and S_1 contains all points with x value *less than* the median x-value.

Can any point in S_0 be dominated by a point in S_1?

Solution: No. From the definition, every point in S_0 has larger x-value than any point in S_1 and thus cannot be dominated by a point in S_1.

(c) (**6 points**) Briefly describe an $O(n \log n)$ time algorithm for finding the set of undominated points. Briefly justify why your algorithm is correct and prove that it runs in the required time.

Solution:

Finishing Key idea: Note that any point in S_1 is dominated if its y value is less than B since every point in S_0 has larger x value than all the points in S_1.

Find the median (using the algorithm from class), split the points into S_0 and S_1. Recursively find the undominated points S_0' and S_1' in S_0 and S_1 respectively, then remove the points in S_1' whose y value is smaller than B, the maximum y-value of any point in S_0.

The runtime is $T(n) = 2T(n/2) + O(n)$ since the median can be found in $O(n)$ time, the partitioning can be done with a scan in $O(n)$ time, and the removing the dominated points returned in the recursive call on S_1 can be done in $O(n)$ time.

Alternative Solution: Sort the points in decreasing order of x value. Then scan the list, and keep any point that attains a new maximal y value. This works because any point is be dominated by an earlier

(higher x valued) point with higher y value, and can only be dominated by such points. The sorting takes $O(n \log n)$ time.

9 Holiday planning

(10 points)

Alice and Bob are childhood friends, but they now go to different colleges in different cities. Alice lives in city t and Bob in City s. Spring break is coming, and Bob plans to fly to see Alice.

The list of possible flights can be described as a directed graph $G = (V, E)$, and $\{w_e | e \in E\}$ represent the cost of every single flight. As a college student, Bob wants to keep the traveling expense as low as possible. He has c universal coupons, each of which can be used for a 50% discount on any single flight.

Define a shortest path problem whose solution can be used to find the route from s to t with minimal spending. Precisely describe the set of vertices and edges in your graph and which shortest path algorithm you will use, and what you output from that computation. Briefly justify that your solution produces the correct answer, and state its running time in terms of $|V|$, $|E|$ and c.

Solution:

A single execution of Dijkstra on G and minus the top c flights on the shortest path won't work correctly. We need to keep track of how many coupons we have used on the fly.

Make $c+1$ copies of the graph and stack them together (indexed from 0 to c). Vertices and edges of layer l are represented by v_i^l and e_j^l ($1 \leq i \leq |V|$ and $1 \leq j \leq |E|$). If two cities are directly connected in G, then add an edge between all adjacent layers in G'. Specifically, If v_i and v_j are connected, then we add v_i^l and v_j^{l+1} with weight $\frac{1}{2}w_{<i,j>}$ for $0 \leq l \leq c-1$. Arriving at v_i^l means that you are at city i and have used l coupons. Every time we go to a higher layer, we take a flight for half its price at the cost of a coupon.

We can run Dijkstra on G' starting from s^0. Since it is possible that the whole trip takes less than c flights, the most economic plan takes $\min_l d_t^l$. Alternatively, we can add a single dummy destination and connect city t of all layers to it with weight 0. This gives the same time complexity and the possiblity of early stopping (When one of $l+1$ destinations is reached).

In $G' = (V', E')$, we have $|V'| = (c+1)|V|$ and $|E'| = (2c+1)|E|$. The runtime of Dijkstra on such a graph is $O(c(V+E)\log V)$.