CECS 528, Exam 2, Yellow, Fall 2023, Dr. Ebert

NO NOTES, BOOKS, ELECTRONIC DEVICES, OR INTERPERSONAL COMMUNICATION ALLOWED. Submit **AT MOST SIX** solutions. Make sure your name and SID are on each answer sheet and please **USE BOTH SIDES** of each answer sheet to save paper.

Problems (25 Points Each)

1. Answer the following with regards to a correctness-proof outline for the Task Selection algorithm (TSA). Note: correctly solving this problem counts for passing LO5.

   (a) Let $T = t_1, \ldots, t_m$ be the set of non-overlapping tasks selected by TSA and sorted by finish time, i.e. $f(t_i) < f(t_{i+1})$ for all $i = 1, \ldots, m-1$. Let T_{opt} be an optimal set of tasks and assume that, for some $k \geq 1$, $t_1, \ldots, t_{k-1} \in T_{\text{opt}}$, but $t_k \notin T_{\text{opt}}$. Explain why there must be at least one task $t' \in T_{\text{opt}}$ that overlaps with t_k. Hint: "Because if there was no such task ...". (7 pts)

   (b) Explain why there is *at most* one task $t' \in T_{\text{opt}}$ that overlaps with t_k. Hint: assume there are two overlapping tasks, t' and t'', and explain why this creates a contradiction. (10 pts)

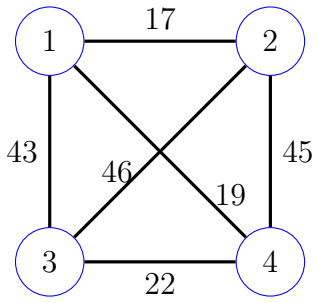
   (c) Thus, we can define a new optimal set of tasks $\hat{T}_{\text{opt}} = T_{\text{opt}} - \{t'\} + \{t_k\}$ that contains $t_1, \ldots, t_k$. Continuing in this manner, we may obtain an optimal set of tasks T_{opt} for which $T \subseteq T_{\text{opt}}$. Moreover, we also have $T_{\text{opt}} \subseteq T$, since there is no way of add another task to T that does not overlap with one of T's tasks. For example, explain why it would be impossible for S_{opt} to possess a task not in S and whose finish time occurs before the start time of any task in S. (8 pts)

2. Do the following. Note: correctly solving this problem counts for passing LO7.

   (a) The dynamic-programming algorithm that solves the `Runaway Traveling Salesperson` optimization problem (Exercise 30 from the Dynamic Programming Lecture) defines a recurrence for the function $\text{mc}(i, A)$. In words, what does $\text{mc}(i, A)$ equal? Hint: do *not* write the recurrence (see Part b). Hint: we call it "Runaway TSP" because the salesperson does *not* return home. (5 pts)

   (b) Provide the dynamic-programming recurrence for $\text{mc}(i, A)$. (5 pts)

   (c) Apply the recurrence from Part b to the graph below in order to calculate $\text{mc}(1, \{2, 3, 4\})$ Show all the necessary computations and use the solutions to compute an optimal path for the salesperson. (15 pts)



3. Part a refers to the original 2SAT algorithm that makes oracle queries, while Part b refers to the improved 2SAT algorithm. Do the following. Note: correctly solving this problem counts for passing LO8.

   (a) Suppose you have been given an unsatisfiable instance $\mathcal{C}$ of 2SAT that consists of 336 variables and 612 binary clauses. If you apply the original 2SAT algorithm to $\mathcal{C}$, what is the *worst case* number of oracle queries that will have be made before concluding that $\mathcal{C}$ is unsatisfiable? Explain. (8 pts)

   (b) Consider the 2SAT instance

   $$\mathcal{C} = \{(\overline{x}_1, x_4), (x_1, \overline{x}_5), (\overline{x}_2, \overline{x}_3), (\overline{x}_2, x_4), (x_2, x_6), (x_3, x_4), (\overline{x}_3, x_6), (\overline{x}_4, \overline{x}_5), (\overline{x}_4, x_5)\}.$$

   Draw the implication graph $G_{\mathcal{C}}$. (5 pts)

   (c) For the 2SAT instance of part b, find a literal l for which i) R_l is an inconsistent reachability set, ii) $R_{\overline{l}}$ is a consistent reachability set, and iii) $\alpha_{R_{\overline{l}}}$ satisfies *all* the clauses of $\mathcal{C}$. For full credit clearly state the literal l you have chosen and verify that each of the three properties are satisfied. (12 pts)

4. Consider the following greedy algorithm for finding a minimum spanning tree within a connected weighted graph $G = (V, E)$. Sort the edges by *decreasing* order of weight. For each edge e in the sorted order, remove e from G if G remains connected after e has been removed. Otherwise, keep e in G. Claim: once G has only $|V| - 1 = n - 1$ edges remaining, then it will represent a minimum spanning tree. The following is a proof outline of this fact.

   (a) Let $R = e_1, \ldots, e_r$ be the r edges that were removed by the algorithm and in the order that they were removed. Let R_{opt} be an optimal set of removed edges. In other words, $E - R_{\text{opt}}$ yields an mst. Let k be the least index for which $e_k \notin R_{\text{opt}}$, i.e. $e_1, \ldots, e_{k-1} \in R_{\text{opt}}$, but not e_k. Consider the set $R_{\text{opt}} + e_k$, where we've removed e_k from the mst and added it to R_{opt}, which results in the mst being broken into two disconnected subgraphs. Explain why there must be at least one edge $e \in R_{\text{opt}}$ such that, i) $e \neq e_i$ for each $i = 1, \ldots, k-1$, and ii) if we remove e from R_{opt} and add it back to G, then G will once again be an mst. (10 pts)

   (b) Moreover, explain why the edge $e \in R_{\text{opt}}$, whose existence we've established from part a, must come *after* e_k in the sorted order. (10 pts)

   (c) Finally, explain why $E - R_{\text{opt}} + e_k - e$ is also an mst. (5 pts)

5. Consider the problem of counting the number of times a bit string u appears in a bit string v, where we assume u's bits do *not* have to appear consecutively. For example, if $u = 011$ and $v = 001011$, then u appears seven times in v at the following index locations:

   $$(135), (136), (156), (235), (236), (256), \text{ and } (456).$$

   Let $N(i, j)$ denote the number of times u-prefix $u_1 \cdots u_i$ appears in v-prefix $v_1 \cdots v_j$. We assume that $N(0, j) = N(i, 0) = 0$.

   (a) Provide a dynamic-programming recurrence for $N(i, j)$. Hint: an appearance of the u-prefix in the v-prefix may or may not make use of bit j of the v-prefix. (18 pts)

(b) Apply your recurrence to the problem instance $u = 101$ and $v = 1101011$. Provide the matrix of subproblem solutions. (7 pts)

6. Prove that a tree with n vertices has exactly $n - 1$ edges. Hint: you may assume that every tree has at least one degree-1 vertex. (25 pts)

LO Makeup Problems (0 Points Each)

LO1. Solve the following problems.

(a) Compute the multiplicative inverse of 28 modulo 87.

(b) For the Strassen-Solovay primality test, verify that $a = 2$ an accomplice to $n = 5$ being a prime number. Show all work.

LO2. Solve the following problems.

(a) Use the Master Theorem to determine the growth of $T(n)$ if it satisfies the recurrence $T(n) = 5T(n/4) + n \log^4 n$. Defend your answer.

(b) Use the substitution method to prove that, if $T(n)$ satisfies

$$T(n) = 4T(n/2) + n \log n,$$

Then $T(n) = \Omega(n^2)$.

LO3. Solve the following problems.

(a) Consider the following algorithm called `multiply` for multiplying two n-bit binary numbers x and y. Assuming n is even, let x_L and x_R be the leftmost $n/2$ and rightmost $n/2$ bits of x respectively. Define y_L and y_R similarly. Let P_1 be the result of calling `multiply` on inputs x_L and y_L, P_2 be the result of calling `multiply` on inputs x_R and y_R, and P_3 the result of calling `multiply` on inputs $x_L + x_R$ and $y_L + y_R$. Then return the value $P_1 \times 2^n + (P_3 - P_1 - P_2) \times 2^{n/2} + P_2$. Explain in detail why the algorithm's running time satisfies $T(n) = 3T(n/2) + n$.

(b) Using the `multiply` algorithm from part a, and for the two binary integers $x = 10100101$ and $y = 11110000$, determine the values of P_1, P_2, and P_3 at the root level of recursion, and verify that $xy = P_1 \times 2^n + (P_3 - P_1 - P_2) \times 2^{n/2} + P_2$. Hint: you may evaluate P_1, P_2, and P_3 non-recursively using base-10.

LO4. Answer/solve the following.

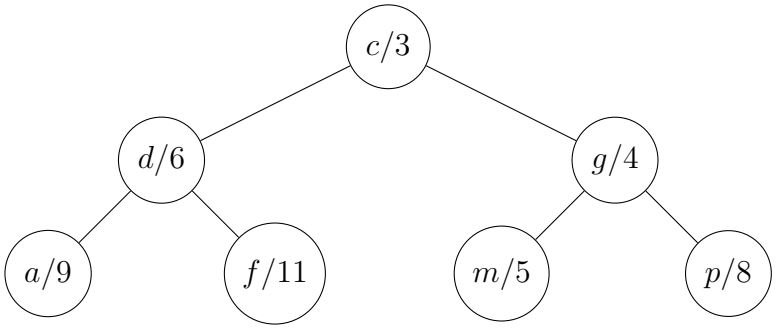
(a) The FFT algorithm owes its existence to what two properties that are possessed by the nth roots of unity when n is even?

(b) Compute $\mathrm{DFT}_4^{-1}(7, 3, -2, 5)$ using the IFFT algorithm. Show the solution to each of the seven subproblem instances and, for each one, clearly represent it using DFT^{-1} notation and apply the formula for computing it. Show all work.

LO6. The tree below shows the state of the binary min-heap at the beginning of some round of Prim's algorithm, applied to some weighted graph G. If G has edges

$$(b, c, 3), (c, e, 7), (c, f, 3), (c, g, 6), (c, p, 2),$$

then draw a plausible state of the heap at the end of the round.



Solutions to 25-Point Problems

1. Answer the following with regards to a correctness-proof outline for the Task Selection algorithm (TSA). Note: correctly solving this problem counts for passing LO5.

   (a) Let $T = t_1, \ldots, t_m$ be the set of non-overlapping tasks selected by TSA and sorted by finish time, i.e. $f(t_i) < f(t_{i+1})$ for all $i = 1, \ldots, m-1$. Let T_{opt} be an optimal set of tasks and assume that, for some $k \geq 1$, $t_1, \ldots, t_{k-1} \in T_{\text{opt}}$, but $t_k \notin T_{\text{opt}}$. Explain why there must be at least one task $t' \in T_{\text{opt}}$ that overlaps with t_k. Hint: "Because if there was no such task ...". (7 pts)

   Solution. Because if there was no such task, then one could add t_k to T_{opt} and obtain a better solution, which contradicts T_{opt} being optimal.

   (b) Explain why there is *at most* one task $t' \in T_{\text{opt}}$ that overlaps with t_k. Hint: assume there are two overlapping tasks, t' and t'', and explain why this creates a contradiction. (10 pts)

   Solution. If two tasks t' and t'' from T_{opt} overlapped with t_k, then one of the two, say t' would have a finish time that comes before the finish time of t_k (why ?). Moreover, since $t_{k-1} \in T_{\text{opt}}$, t' would start at or after t_{k-1}. Thus, in round k the algorithm would have selected t' instead of t_k.

   (c) Thus, we can define a new optimal set of tasks $\hat{T}_{\text{opt}} = T_{\text{opt}} - \{t'\} + \{t_k\}$ that contains $t_1, \ldots, t_k$. Continuing in this manner, we may obtain an optimal set of tasks T_{opt} for which $T \subseteq T_{\text{opt}}$. Moreover, we also have $T_{\text{opt}} \subseteq T$, since there is no way of add another task to T that does not overlap with one of T's tasks. For example, explain why it would be impossible for S_{opt} to possess a task not in S and whose finish time occurs before the start time of any task in S. (8 pts)

   Solution. The first task t_1 selected by the `TSA` algorithm is the task that finishes the earliest. Thus, there cannot be a task that starts before t_1.

2. Do the following. Note: correctly solving this problem counts for passing LO7.

   (a) The dynamic-programming algorithm that solves the `Runaway Traveling Salesperson` optimization problem (Exercise 30 from the Dynamic Programming Lecture) defines a recurrence for the function $\text{mc}(i, A)$. In words, what does $\text{mc}(i, A)$ equal? Hint: do *not* write the recurrence (see Part b). Hint: we call it "Runaway TSP" because the salesperson does *not* return home. (5 pts)

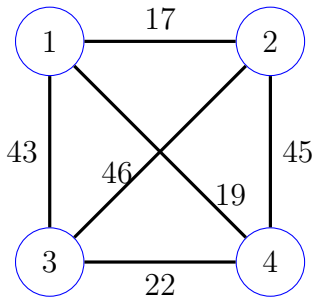
   Solution. $\text{mc}(i, A)$ equals the cost of the minimum-cost simple path that starts at i and visits every vertex in A.

   (b) Provide the dynamic-programming recurrence for $\text{mc}(i, A)$. (5 pts)

   Solution. We have

$$\text{mc}(i, A) = \begin{cases} 0 & \text{if } A = \emptyset \\ C_{ij} & \text{if } A = \{j\} \\ \min\limits_{j \in A}(C_{ij} + \text{mc}(j, A - \{j\}) & \text{otherwise} \end{cases}$$

(c) Apply the recurrence from Part b to the graph below in order to calculate $\text{mc}(1, \{2, 3, 4\})$ Show all the necessary computations and use the solutions to compute an optimal path for the salesperson. (15 pts)



Solution. Start with $\text{mc}(1, \{2, 3, 4\})$ and proceed to compute other mc values as needed.

$$\text{mc}(1, \{2, 3, 4\}) = \min(17 + \text{mc}(2, \{3, 4\}), 43 + \text{mc}(3, \{2, 4\}), 19 + \text{mc}(4, \{2, 3\})).$$

$$\text{mc}(2, \{3, 4\}) = \min(46 + \text{mc}(3, \{4\}), 45 + \text{mc}(4, \{3\})) = \min(46 + 22, 45 + 22) = 67.$$

$$\text{mc}(3, \{2, 4\}) = \min(46 + \text{mc}(2, \{4\}), 22 + \text{mc}(4, \{2\})) = \min(46 + 45, 22 + 45) = 67.$$

$$\text{mc}(4, \{2, 3\}) = \min(45 + \text{mc}(2, \{3\}), 22 + \text{mc}(3, \{2\})) = \min(45 + 46, 22 + 46) = 68.$$

Therefore,

$$\text{mc}(1, \{2, 3, 4\}) = \min(17 + \text{mc}(2, \{3, 4\}), 43 + \text{mc}(3, \{2, 4\}), 19 + \text{mc}(4, \{2, 3\})) =$$

$$\min(17 + 67, 43 + 67, 19 + 68) = 84.$$

This gives the optimal path $P = 1, 2, 4, 3$.

3. Part a refers to the original `2SAT` algorithm that makes oracle queries, while Part b refers to the improved `2SAT` algorithm. Do the following. Note: correctly solving this problem counts for passing LO8.

(a) Suppose you have been given an unsatisfiable instance $\mathcal{C}$ of `2SAT` that consists of 336 variables and 612 binary clauses. If you apply the original `2SAT` algorithm to $\mathcal{C}$, what is the *worst case* number of oracle queries that will have be made before concluding that $\mathcal{C}$ is unsatisfiable? Explain. (8 pts)

Solution. Worst case is that $2 \times 336 = 672$ queries have to be made. This would occur in the case that the only variable that is responsible for an inconsistent is the 336th variable checked.

(b) Consider the 2SAT instance

$$\mathcal{C} = \{(\overline{x}_1, x_4), (x_1, \overline{x}_5), (\overline{x}_2, \overline{x}_3), (\overline{x}_2, x_4), (x_2, x_6), (x_3, x_4), (\overline{x}_3, x_6), (\overline{x}_4, \overline{x}_5), (\overline{x}_4, x_5)\}.$$

Draw the implication graph $G_{\mathcal{C}}$. (5 pts)

Solution. Please see

`https://home.csulb.edu/~tebert/teaching/fall22/528/assess/LO9-11-09-2022/LO9-11-09-2022-Solution-LO9.pdf`

(c) For the 2SAT instance of part b, find a literal l for which i) R_l is an inconsistent reachability set, ii) $R_{\overline{l}}$ is a consistent reachability set, and iii) $\alpha_{R_{\overline{l}}}$ satisfies *all* the clauses of $\mathcal{C}$. For full credit clearly state the literal l you have chosen and verify that each of the three properties are satisfied. (12 pts)

Solution. Please see

`https://home.csulb.edu/~tebert/teaching/fall22/528/assess/LO9-11-09-2022/LO9-11-09-2022-Solution-LO9.pdf`

4. Consider the following greedy algorithm for finding a minimum spanning tree within a connected weighted graph $G = (V, E)$. Sort the edges by *decreasing* order of weight. For each edge e in the sorted order, remove e from G if G remains connected after e has been removed. Otherwise, keep e in G. Claim: once G has only $|V| - 1 = n - 1$ edges remaining, then it will represent a minimum spanning tree. The following is a proof outline of this fact.

(a) Let $R = e_1, \ldots, e_r$ be the r edges that were removed by the algorithm and in the order that they were removed. Let R_{opt} be an optimal set of removed edges. In other words, $T_{\text{opt}} = E - R_{\text{opt}}$ yields an mst. Let k be the least index for which $e_k \notin R_{\text{opt}}$, i.e. $e_1, \ldots, e_{k-1} \in R_{\text{opt}}$, but not e_k. Consider the set $R_{\text{opt}} + e_k$, where we've removed e_k from the mst and added it to R_{opt}, which results in the mst being broken into two disconnected subgraphs. Explain why there must be at least one edge $e \in R_{\text{opt}}$ such that, i) $e \neq e_i$ for each $i = 1, \ldots, k$, and ii) if we remove e from R_{opt} and add it back to G, then G will once again be an mst. (10 pts)

Solution. Let A and B denote the two trees of the forest that forms when e_k is removed from the mst T_{opt}. When the algorithm removed e_k, G remained connected. Therefore, at that time there must have been some edge e in G for which e was incident with a vertex of A and with a vertex of B. Otherwise, removal of e_k would have disconnected G. Moreover, this edge $e \in R_{\text{opt}}$ since otherwise the mst would have a cycle that includes both e_k and e. Finally $e \neq e_i$ for each $i = 1, \ldots, k$ since e remained in G after each e_i was removed.

(b) Moreover, explain why the edge $e \in R_{\text{opt}}$, whose existence we've established from part a, must come *after* e_k in the sorted order. (10 pts)

Solution. As was mentioned in part a, edge e was in G when e_k was removed, and thus it must come after e_k in the sorted order. Otherwise, it would have been successfully

removed (since e_k would still be in G) and would not have still been in G when e_k was removed.

(c) Finally, explain why $E - R_{\text{opt}} + e_k - e$ is also an mst. (5 pts)

Solution. Since e comes after e_k in the sorted order, $w(e) \leq w(e_k)$, and replacing e_k with e in the mst results in a tree whose cost is no greater than the original one. Therefore, the modified tree is also an mst.

5. Consider the problem of counting the number of times a bit string u appears in a bit string v, where we assume u's bits do *not* have to appear consecutively. For example, if $u = 011$ and $v = 001011$, then u appears seven times in v at the following index locations:

$$(135), (136), (156), (235), (236), (256), \text{ and } (456).$$

Let $N(i, j)$ denote the number of times u-prefix $u_1 \cdots u_i$ appears in v-prefix $v_1 \cdots v_j$. We assume that $N(0, j) = 0$ and $N(i, 0) = 0$.

(a) Provide a dynamic-programming recurrence for $N(i, j)$. Hint: an appearance of the u-prefix in the v-prefix may or may not make use of bit j of the v-prefix. (18 pts)

Solution. We have

$$N(i,j) = \begin{cases} 0 & \text{if } i > j \\ u[1:i] = v[1:j] & \text{if } i = j \\ \sum_{k=1}^{j} (u[1] = v[k]) & \text{if } i = 1 \\ N(i, j-1) & \text{if } u[i] \neq v[j] \\ N(i, j-1) + N(i-1, j-1) & \text{if } u[i] = v[j] \end{cases}$$

(b) Apply your recurrence to the problem instance $u = 101$ and $v = 1101011$. Provide the matrix of subproblem solutions. (7 pts)

Solution.

i/j	1	1	0	1	0	1	1
1	1	2	2	3	3	4	5
0	0	0	2	2	5	5	5
1	0	0	0	2	2	7	12

6. Prove that a tree with n vertices has exactly $n - 1$ edges. Hint: you may assume that every tree has at least one degree-1 vertex. (25 pts)

Solution. See solution to Exercise 2 of the Greedy Graph Algorithms lecture.

Solutions to LO Makeup Problems

LO1. Solve the following problems.

(a) Compute the multiplicative inverse of 28 modulo 87.

Solution. 28 is its own inverse modulo 87.

(b) For the Strassen-Solovay primality test, verify that $a = 2$ an accomplice to $n = 5$ being a prime number. Show all work.

Solution. $2^{(5-1)/2} = 2^2 \equiv 4 \equiv -1 \bmod 5$. Also $\left(\frac{2}{5}\right) = -1$ since $5 \equiv -3 \bmod 8$.

LO2. Solve the following problems.

(a) Use the Master Theorem to determine the growth of $T(n)$ if it satisfies the recurrence $T(n) = 5T(n/4) + n\log^4 n$. Defend your answer.

Solution. Since $\log_4 5 > 1$, $n\log^4 n = \mathrm{O}(n^{\log_4 5-\epsilon})$ for ϵ sufficiently small, and it follows by Case 1 of the Master Theorem that $T(n) = \Theta(n^{\log_4 5})$.

(b) Use the substitution method to prove that, if $T(n)$ satisfies

$$T(n) = 4T(n/2) + n\log n,$$

Then $T(n) = \Omega(n^2)$.

Solution. Inductive Assumption: $T(k) \geq Ck^2$ for all $k < n$. Show $T(n) \geq Cn^2$. We have

$$T(n) = 4T(n/2) + n\log n \geq 4C(\frac{n}{2})^2 + n\log n = Cn^2 + n\log n \geq Cn^2 \iff n\log n \geq 0$$

which is true for $n \geq 1$. Therefore, $T(n) = \Omega(n^2)$.

LO3. Solve the following problems.

(a) Consider the following algorithm called `multiply` for multiplying two n-bit binary numbers x and y. Assuming n is even, let x_L and x_R be the leftmost $n/2$ and rightmost $n/2$ bits of x respectively. Define y_L and y_R similarly. Let P_1 be the result of calling `multiply` on inputs x_L and y_L, P_2 be the result of calling `multiply` on inputs x_R and y_R, and P_3 the result of calling `multiply` on inputs $x_L + x_R$ and $y_L + y_R$. Then return the value $P_1 \times 2^n + (P_3 - P_1 - P_2) \times 2^{n/2} + P_2$. Explain in detail why the algorithm's running time satisfies $T(n) = 3T(n/2) + n$.

Solution. The divide-and-conquer algorithm creates three subproblems, each with size $n/2$. Moreover, it requires $\mathrm{O}(n)$ steps to form the instances of P_3 and $\mathrm{O}(n)$ steps to compute the expression $P_1 \times 2^n + (P_3 - P_1 - P_2) \times 2^{n/2} + P_2$. This is because multiplying by, e.g., 2^n requires shifting the bits of P_1 to the left by n places which takes $\mathrm{O}(n)$ steps. The same is true for multiplying with $2^{n/2}$. Finally, the additions required to get the final answer all require $\mathrm{O}(n)$ steps since the numbers being added all have $\mathrm{O}(n)$ bits.

(b) Using the `multiply` algorithm from part a, and for the two binary integers $x = 10100101$ and $y = 11110000$, determine the values of P_1, P_2, and P_3 at the root level of recursion, and verify that $xy = P_1 \times 2^n + (P_3 - P_1 - P_2) \times 2^{n/2} + P_2$. Hint: you may evaluate P_1, P_2, and P_3 non-recursively using base-10.

Solution. $x_L = 10$, $x_R = 5$, $y_L = 15$, $y_R = 0$, $P_1 = 150$, $P_2 = 0$, $P_3 = 225$, $xy = 39,600$.

LO4. Answer/solve the following.

(a) The FFT algorithm owes its existence to what two properties that are possessed by the nth roots of unity when n is even?

Solution. The squares of the nth roots of unity give the $n/2$ roots of unity, and the nth roots of unity come in additive inverse pairs so that their squares produce only $n/2$ values (which happen to be the $n/2$ roots of unity).

(b) Compute $\mathrm{DFT}_4^{-1}(7, 3, -2, 5)$ using the IFFT algorithm. Show the solution to each of the seven subproblem instances and, for each one, clearly represent it using DFT^{-1} notation and apply the formula for computing it. Show all work.

Solution. We have

$$\mathrm{DFT}_1^{-1}(7) = 7 \text{ and } \mathrm{DFT}_1^{-1}(-2) = -2.$$

$$\mathrm{DFT}_2^{-1}(7, -2) = 1/2((7, 7) + (1, -1) \odot (-2, -2)) = (5/2, 9/2).$$

$$\mathrm{DFT}_1^{-1}(3) = 3 \text{ and } \mathrm{DFT}_1^{-1}(-5) = -5.$$

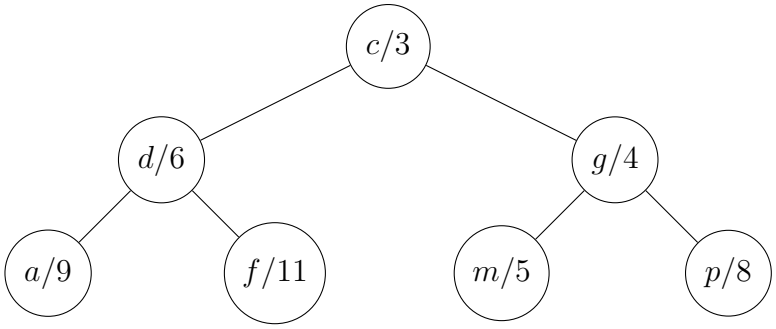
$$\mathrm{DFT}_2^{-1}(3, -5) = 1/2((3, 3) + (1, -1) \odot (-5, -5)) = (-1, 4).$$

$$\mathrm{DFT}_4^{-1}(7, 3, -2, -5) = 1/2((5/2, 9/2, 5/2, 9/2) + (1, -i, -1, i) \odot (-1, 4, -1, 4)) =$$
$$\frac{1}{4}(3, 9 - 8i, 7, 9 + 8i).$$

LO6. The tree below shows the state of the binary min-heap at the beginning of some round of Prim's algorithm, applied to some weighted graph G. If G has edges

$$(b, c, 3), (c, e, 7), (c, f, 3), (c, g, 6), (c, p, 2),$$

then draw a plausible state of the heap at the end of the round.



Solution.