Section 07: Solutions

Section Problems

1. In-Depth Recurrence

Consider the following recurrence: $A(n) = \begin{cases} 1 & \text{if } n \leq 1 \\ 3A(n/6) + n & \text{otherwise} \end{cases}$

We want to find an *exact* closed form of this equation by using the tree method. Suppose we draw out the total work done by this method as a tree, as discussed in lecture. Let n be the initial input to A.

(a) What is the size of the input at level i (as in class, call the root level 0)?

Solution:

We divide by 6 at each level, so the input size is $n/6^i$.

(b) What is the number of nodes at level i?

Note: let $i = 0$ indicate the level corresponding to the root node. So, when $i = 0$, your expression should be equal to 1.

Solution:

Each (non-base-case) node produces 3 more nodes, so at level i we have 3^i nodes.

(c) What is the total work at the i^{th} **recursive** level?

Solution:

Combining our last two parts: $3^i \cdot \frac{n}{6^i} = \frac{n}{2^i}$

(d) What is the last level of the tree?

Solution:

We hit our base case when $n/6^i = 1$, which is at level $i = \log_6(n)$.

(e) What is the work done in the base case?

Solution:

From previous parts, there are $3^{\log_6(n)}$ nodes at that level, from the recurrence each does 1 unit of work, so we get $1 \cdot 3^{\log_6(n)}$ work.

(f) Combine your answers from previous parts to get an expression for the total work.

Solution:

We know the expression for the work done at each recursive level and the base case level. Combining these we have:

$$\sum_{i=0}^{\log_6(n)-1} \frac{n}{2^i} + 3^{\log_6(n)}$$

Side note: Yes, it says $\log_6(n) - 1$ and not $\log_6(n)$. This is because the last level ($\log_6(n)$) is the base case level and has a different formula.

(g) Simplify to a closed form.

Note: you do not need to simplify your answer, once you found the closed form. Hint: You should use the finite geometric series identity somewhere while finding a closed form.

Solution:

We combine all the pieces and simplify:

$$\begin{aligned} A(n) &= \sum_{i=0}^{\log_6(n)-1} \frac{n}{2^i} + 3^{\log_6(n)} \\ &= n \sum_{i=0}^{\log_6(n)-1} \left(\frac{1}{2}\right)^i + 3^{\log_6(n)} \end{aligned}$$

We'll apply two identities: to the summation we apply the finite geometric series identity; and to the base-case work, we apply the power of a log identity. We get:

$$A(n) = n \cdot \frac{\left(\frac{1}{2}\right)^{\log_6(n)} - 1}{\frac{1}{2} - 1} + n^{\log_6(3)}$$

You don't have to simplify further, but if you were to simplify, you would get:

$$\begin{aligned} A(n) &= n \cdot \frac{\left(\frac{1}{2}\right)^{\log_6(n)} - 1}{1/2 - 1} + n^{\log_6(3)} \\ &= -2n\left(n^{\log_6(1/2)} - 1\right) + n^{\log_6(3)} \\ &= -2n\left(n^{\log_6(3/6)} - 1\right) + n^{\log_6(3)} \\ &= -2n\left(n^{\log_6(3)-\log_6(6)} - 1\right) + n^{\log_6(3)} \\ &= -2n\left(\frac{n^{\log_6(3)}}{n} - 1\right) + n^{\log_6(3)} \\ &= -2n^{\log_6(3)} + 2n + n^{\log_6(3)} \\ &= 2n - n^{\log_6(3)} \end{aligned}$$

(h) Use the master theorem to find a big-Θ bound of $A(n)$.

Solution:

We check that $\log_6(3) < 1$, so we get $A(n) \in \Theta(n)$, which is consistent with our answer in the last part.

2. Recurrences

For each of the following recurrences, find their closed form using the tree method. Then, check your answer using the master method (if applicable). It may be a useful guide to use the steps from part 2 of this handout to help you with all the parts of solving a recurrence problem fully.

(a) $J(k) = \begin{cases} 1 & \text{if } k = 1 \\ 5J(k/5) + k^3 & \text{otherwise} \end{cases}$

Solution:

Given this recurrence, we know...

- Size of input at level i is $k/5^i$
- Number of nodes on level i: 5^i
- Total work done per (recursive) level i: $5^i \cdot \frac{k^3}{5^{3i}} = 5^i \cdot \frac{k^3}{125^i}$
- Last level of the tree: $\log_5(k)$
- Total work done in base case: $1 \cdot 5^{\log_5(k)}$

So we get the expression:

$$\left(\sum_{i=0}^{\log_5(k)-1} 5^i \cdot \frac{k^3}{125^i} \right) + 5^{\log_5(k)}$$

We apply the finite geometric series to get:

$$k^3 \frac{\left(\frac{5}{125}\right)^{\log_5(k)} - 1}{\frac{5}{125} - 1} + 5^{\log_5(k)}$$

If we wanted to simplify, we'd get:

$$\frac{25k^3}{24} - \frac{k}{24}$$

We can also apply the master theorem here. Note that $\log_b(a) = \log_5(5) = 1 < 3 = c$, so we know $J(k) \in \Theta\left(k^3\right)$.

(b) $S(q) = \begin{cases} 1 & \text{if } q = 1 \\ 2S(q-1)+1 & \text{otherwise} \end{cases}$

Solution:

Given this recurrence, we know...

- Size of input at level i is $q - i$
- Number of nodes on level i: 2^i
- Total work done at (recursive) level i: $2^i \cdot 1$
- Last level of the tree: $q - 1$
- Total work done in base case: $1 \cdot 2^{q-1}$

Note that these expressions look a little different from the ones we've seen up above. This is because we aren't *dividing* our terms by some constant factor – instead, we're *subtracting* them.

So we get the expression:

$$\left(\sum_{i=0}^{q-1-1} 2^i \right) + 2^{q-1}$$

We apply the finite geometric series to get:

$$\frac{2^{q-1} - 1}{2 - 1} + 2^{q-1}$$

If we wanted to simplify, we'd get:

$$2^q - 1$$

Note that we may NOT apply the master theorem here – our original recurrence doesn't match the form given in the theorem.

(c) $Z(x) = \begin{cases} \log(x) & \text{if } x \leq 9 \\ 3Z(x/3) + 1 & \text{otherwise} \end{cases}$

Solution:

Given this recurrence, we know...

- Size of input at level i: $x/3^i$
- Number of nodes on level i: 3^i
- Total work done per (recursive) level i: 3^i
- Last level of the tree: $\log_3(x/9) = \log_3(x) - 2$
- Total work done in base case: $\log_2(9) \cdot 3^{\log_3(x)-2}$

Note that the height here is different, since the recursive function hits the base case when $x \leq 9$.

So we get the expression:

$$\left(\sum_{i=0}^{\log_3(x)-2-1} 3^i \right) + \log_2(9) \cdot 3^{\log_3(x)-2}$$

We apply the finite geometric series to get:

$$\frac{3^{\log_3(x)-1} - 1}{3-1} + \log_2(9) 3^{\log_3(x)-2}$$

If we wanted to simplify, we'd get:

$$\frac{x}{6} - \frac{1}{2} + \log_2(9)\frac{x}{9} = \frac{2\log_2(9)+3}{18} \cdot x - \frac{1}{2}$$

Initially, it doesn't seem like we can apply the master theorem because the base case doesn't match the exact form of the theorem.

However, since $x \leq 9$ in the base case, $\log(x)$ always ends up being a constant, so Master Theorem actually applies.

Note that $\log_b(a) = \log_3(3) = 1 > 0 = c$, so we know $Z(x) \in \Theta\left(x^{\log_b(a)}\right)$. This ends up being $Z(x) \in \Theta\left(x^{\log_3(3)}\right)$ which simplifies to just $Z(x) \in \Theta(x)$.

(d) $T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n/2) + 3 & \text{otherwise} \end{cases}$

Solution:

Given this recurrence, we know...

- Size of input at level i: $n/2^i$
- Number of nodes on level i: $1^i = 1$
- Total work done per (recusive) level i: $1 \cdot 3 = 3$
- Last level of the tree: $\log_2(n)$
- Total work done in base case: $1 \cdot 1^{\log_3(n)} = 1$

So we get the expression:

$$\left(\sum_{i=0}^{\log_2(n)-1} 3 \right) + 1$$

Using the summation of a constant identity, we get:

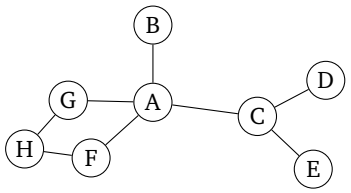
$$3\log_2(n) + 1$$

We can apply the master thereom here. Note that $\log_b(a) = \log_2(1) = 0 = c$, which means that $T(n) \in \Theta\left(n^c \log(n)\right)$ which is $T(n) \in \Theta\left(n^0 \log(n)\right)$ which further simplifies to $T(n) \in \Theta\left(\log(n)\right)$.

This agrees with our simplified form.

3. Graph traversal

(a) Consider the following graph. Suppose we want to traverse it, starting at node A.



If we traverse this using *breadth-first search*, what are *two* possible orderings of the nodes we visit? What if we use *depth-first search*?

Solution:

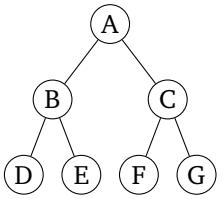
Here are two possible orderings for BFS:

- A, G, F, B, C, H, D, E
- A, C, B, F, G, D, H, E

Here are two possible orderings for DFS:

- A, G, H, F, C, D, E, B
- A, B, C, E, D, F, H, G

(b) Same question, but on this graph:



Solution:

Here are two possible orderings for BFS:

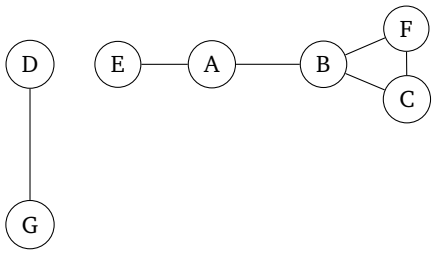
- A, B, C, D, E, F, G
- A, C, B, F, G, D, E

Here are two possible orderings for DFS:

- A, B, D, E, C, F, G
- A, C, G, F, B, E, D

4. Graph properties

(a) Consider the *undirected, unweighted* graph below.



Answer the following questions about this graph:

(i) Find V, E, $|V|$, and $|E|$.

(ii) What is the maximum *degree* of the graph?

(iii) Are there any cycles? If so, where?

(iv) What is the maximum length simple path in this graph?

(v) What is one edge you could add to the graph that would increase the length of the maximum length simple path of the new graph to 6?

(vi) What are the *connected components* of the graph?

Solution:

(i) $V = \{A, B, C, D, E, F, G\}$ and $E = \{(D, G), (E, A), (A, B), (B, F), (F, C), (C, B)\}$. This means that $|V| = 7$ and $|E| = 6$.

(ii) The vertex with the max degree is B, which has a degree of 3.

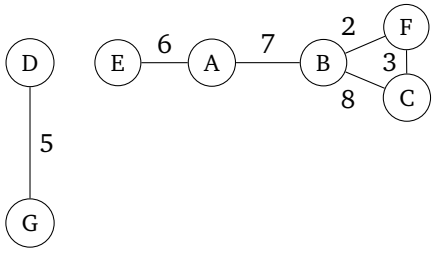
(iii) There is indeed a cycle, between B, C, and F.

(iv) The maximum length simple path is $(E, A), (A, B), (B, F)$.

(v) We could add the edge (D, E).

(vi) One connected component is $\{D, G\}$. Another one is $\{E, A, B, C, F\}$.

(b) Consider the *undirected, weighted* graph below.



Answer the following questions about this graph:

(i) What is the path involving the least number of nodes from E to C? What is its cost?

(ii) What is the minimum cost path from E to C? What is its cost?

(iii) What is the minimum length path from E to C? What is its length?

Solution:

(i) The path with the least number of nodes is $(E, A), (A, B), (B, C)$. The cost is 21.

(ii) The minimum cost path is actually $(E, A), (A, B), (B, F), (F, C)$. The cost is 18.

(iii) The path with the shortest length is $(E, A), (A, B), (B, C)$. The length is 3.

5. Implementing graph searches

(a) Come up with pseudocode to implement *breadth-first search* on a graph, given a starting node and an adjacency list representation of the graph. Is your method recursive or not? What data structures do you use?

Solution:

See the pseudocode given in lecture.

(b) Come up with pseudocode to implement *depth-first search* on a graph, given a starting node and an adjacency list representation of the graph. Is your method recursive or not? What data structures do you use?

Solution:

See the pseudocode given in lecture.

Challenge Problems

6. Recurrences

For each of the following recurrences, find their closed form using the tree method. Then, check your answer using the Master Theorem (if applicable).

(a) $Y(q) = \begin{cases} 1 & \text{if } q = 1 \\ 8T(q/2) + q^3 & \text{otherwise} \end{cases}$

Solution:

Given this recurrence, we know...

- Number of nodes on level i: 8^i
- Total work done per level: $8^i \cdot \frac{q^3}{2^{3i}} = 8^i \cdot \frac{q^3}{8^i}$
- Total number of *recursive* levels: $\log_2(q)$
- Total work done in base case: $8^{\log_2(q)}$

So we get the expression:

$$\left(\sum_{i=0}^{\log_2(q)-1} 8^i \cdot \frac{q^3}{8^i} \right) + 8^{\log_2(q)}$$

While we could apply the finite geometric series identity here, there's actually a simpler approach. Notice that the 8^i term cancels itself out. So, we're left with:

$$\left(\sum_{i=0}^{\log_2(q)-1} q^3 \right) + 8^{\log_2(q)}$$

We can then apply the "summation of a constant" identity to get:

$$q^3 \log_2(q) + 8^{\log_2(q)}$$

We can also apply the master theorem here. Note that $\log_b(a) = \log_2(8) = 3 = c$, so we know $Y(q) \in \Theta\left(q^3 \log(q)\right)$.

(b) $T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 8T(n/2) + 4n^2 & \text{otherwise} \end{cases}$

Solution:

Given this recurrence, we know...

- Number of nodes on level i: 8^i
- Total work done per level: $8^i \cdot 4 \left(\frac{n}{2^i}\right)^2 = 8^i \cdot 4 \cdot \frac{n^2}{4^i}$
- Total number of *recursive* levels: $\log_2(n)$
- Total work done in base case: $8^{\log_2(n)}$

So we get the expression:

$$\left(\sum_{i=0}^{\log_2(n)-1} 8^i \cdot 4 \cdot \frac{n^2}{4^i} \right) + 8^{\log_2(n)}$$

We can simplify by pulling the $4n^2$ out of the summation:

$$4n^2 \left(\sum_{i=0}^{\log_2(n)-1} \frac{8^i}{4^i} \right) + 8^{\log_2(n)}$$

This further simplifies to:

$$4n^2 \left(\sum_{i=0}^{\log_2(n)-1} 2^i \right) + 8^{\log_2(n)}$$

After applying the finite geometric series identity, we get:

$$4n^2 \cdot \frac{2^{\log_2(n)} - 1}{2 - 1} + 8^{\log_2(n)}$$

This is a closed form so we could stop, but if we want a tidy solution, we can continue simplifying:

$$\begin{aligned} T(n) &= 4n^2 \cdot \frac{2^{\log_2(n)} - 1}{2 - 1} + 8^{\log_2 2(n)} \\ &= 4n^2 \cdot \left(2^{\log_2(n)} - 1\right) + 8^{\log_2(n)} \\ &= 4n^2 \cdot \left(n^{\log_2(2)} - 1\right) + n^{\log_2(8)} \\ &= 4n^2 \cdot (n - 1) + n^3 \\ &= 5n^3 - 4n^2 \end{aligned}$$

We can apply the master thereom here. Note that $log_b(a) = \log_2(8) = 3 > 2 = c$, which means that $T(n) \in \Theta\left(n^{\log_b(a)}\right)$ which is $T(n) \in \Theta\left(n^{\log_2 8}\right)$ which in turn simplifies to $T(n) \in \Theta\left(n^3\right)$.

This agrees with our simplified form.

(c) $T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 7T(n/3) + 18n^2 & \text{otherwise} \end{cases}$

Solution:

Given this recurrence, we know...

- Number of nodes on level i: 7^i
- Total work done per level: $7^i \cdot 18 \left(\frac{n}{3^i}\right)^2 = 7^i \cdot 18 \cdot \frac{n^2}{9^i}$
- Total number of *recursive* levels: $\log_3(n)$
- Total work done in base case: $7^{\log_3(n)}$

So we get the expression:

$$\left(\sum_{i=0}^{\log_3(n)-1} 7^i \cdot 18 \cdot \frac{n^2}{9^i} \right) + 7^{\log_3(n)}$$

We can simplify by pulling the $18n^2$ out of the summation:

$$18n^2 \left(\sum_{i=0}^{\log_3(n)-1} \frac{7^i}{9^i} \right) + 7^{\log_3(n)}$$

This is equivalent to:

$$18n^2 \left(\sum_{i=0}^{\log_3(n)-1} \left(\frac{7}{9}\right)^i \right) + 7^{\log_3(n)}$$

After applying the finite geometric series identity, we get:

$$18n^2 \cdot \frac{\left(\frac{7}{9}\right)^{\log_3(n)} - 1}{\frac{7}{9} - 1} + 7^{\log_3(n)}$$

This is a closed form so we could stop, but if we want a tidy solution, we can continue simplifying:

$$\begin{aligned}
T(n) &= 18n^2 \cdot \frac{\left(\frac{7}{9}\right)^{\log_3(n)} - 1}{\frac{7}{9} - 1} + 7^{\log_3(n)} \\
&= 18n^2 \cdot \frac{\left(\frac{7}{9}\right)^{\log_3(n)} - 1}{-\frac{2}{9}} + 7^{\log_3(n)} \\
&= -81n^2 \cdot \left(\left(\frac{7}{9}\right)^{\log_3(n)} - 1 \right) + 7^{\log_3(n)} \\
&= -81n^2 \cdot \left(n^{\log_3(7/9)} - 1 \right) + n^{\log_3(7)} \\
&= -81n^2 \cdot \left(n^{\log_3(7)-2} - 1 \right) + n^{\log_3(7)} \\
&= -81n^2 n^{\log_3(7)-2} + 81n^2 + n^{\log_3(7)} \\
&= -80n^{\log_3(7)} + 81n^2
\end{aligned}$$

We can apply the master thereom here. Note that $\log_b(a) = \log_3(7) < 2 = c$, which means that $T(n) \in \Theta\left(n^c\right)$ which is $T(n) \in \Theta\left(n^2\right)$

This agrees with our simplified form.

7. Divide and conquer

Given an array containing elements of type `E` design an algorithm that finds the **majority element** – that is, an element that appears more then $n/2$ times. If no majority element exists, return `null`.

Your algorithm should run in $\mathcal{O}(n \log(n))$ time (and use only $\mathcal{O}(1)$ extra memory).

Note: the items in the array do **NOT** implement `compareTo`. This means you cannot sort the array!

Challenge: can you find the majority in $\mathcal{O}(n)$ time and $\mathcal{O}(1)$ extra memory?

Solution:

The $\mathcal{O}(n \log(n))$ solution works by first splitting the array into two halves. We recurse on both halves and receive back the majority elements for the two halves (if they exists).

Once we finish recursing, there are four different scenarios:

(a) The two subarrays have the same majority element.

This means, by definition, that element must also be the majority of the full array.

Why is this? Suppose that there are n elements in the overall array. If A is the majority of the left half, then that means that by definition, there must be $> \frac{n}{4}$ occurrences of A on the left. Similarly, if A is the majority on the right, there must be $> \frac{n}{4}$ occurrences there.

Therefore, there must be $> \frac{n}{2}$ occurrences of A overall. So we can just return A without needing to check anything else.

(b) The two subarrays have different majority elements.

In that case, we need to figure out which one is the true majority. We take the majority element from the left and loop over the entire array to figure out how many times it appears. We do the same thing with the majority element from the right. This will take $\mathcal{O}(2n) = \mathcal{O}(n)$ time.

If either of them appear more then $\frac{n}{2}$ time, return that element as the majority. If neither of them appear enough times, return null (or whatever else we're using to indicate that there's no majority).

(c) Only one subarray has a majority; the other doesn't.

We do the same sort of looping thing as before, again in $\mathcal{O}(n)$ time.

(d) Neither subarrays have a majority.

We can automatically give up here, for basically the same reason why we could automatically return in case 1.

We end up doing $2T(n) + n$ work in the worst case in the recursive case, which results in $\mathcal{O}(n \log(n))$.

The $\mathcal{O}(n)$ solution is called "Boyer-Moore majority vote algorithm." The Wikipedia page has a good overview of how the algorithm works: https://en.wikipedia.org/wiki/Boyer%E2%80%93Moore_majority_vote_algorithm