CSC165H1 WINTER 2018: PROBLEM SET 4

BY: JACOB CHMURA, CONOR VEDOVA, ERIC KOEHLI

Date: April 4, 2018.

1. Binary representation and algorithm analysis

(a) Prove that the running time of the algorithm `count` is $\mathcal{O}(n \log n)$.

Proof. Upper bound on the runtime:

Let n be the input value to `count`. For a fixed iteration of the outer loop, the inner loop index `j` is initialized to $\lfloor \log_2 n \rfloor$, which is the last index of the list `bits` and iterates over the list in reverse. Then the inner loop can iterate *at most* $\lfloor \log_2 n \rfloor$ times, with each iteration a single step.

The outer loop iterates n times. Since the inner loop condition is independent of the outer loop counter, to get the total cost, we multiply n by the cost of the inner loop. Then the total cost is at most,

$$n \lfloor \log_2 n \rfloor$$

steps (ignoring constant terms). Since $\log_2 n \geq \lfloor \log_2 n \rfloor$, we conclude that $RT_{count}(n) \in \mathcal{O}(n \log n)$.

□

(b) Prove that the running time of the algorithm `count` is $\Omega(n)$.

Proof. Lower bound on the runtime:

For a fixed iteration of the outer loop, the inner loop runs *at least* zero times. The outer loop iterates n times, and since each iteration takes at least constant time (it doesn't depend on the input size), the total cost is at least n steps. Thus $RT_{count} \in \Omega(n)$.

□

2. Worst-case and best-case algorithm analysis

(a) Prove that $WC(n) \in \mathcal{O}(n)$.

Proof. Upper bound on the worst-case:

We want to show that for any arbitrary list `L` of length n, the runtime of `myprogram` runs in *at most* linear time.

Let `n = len(L)` be the length of the input list, `L`. Also, let $k \in \mathbb{N}$ and let i_k be the value of the index i after k loop iterations. There are two ways in which i can decrement on a given loop iteration, depending on the truth value on `line 6` of the code. The worst case occurs when i decreases the least on each iteration, and since this causes the most iterations of the loop, we want an upper bound on the loop counter i. The smallest change for a given iteration of the loop is that i decreases by 1, and this only occurs if the `else` blocks executes on every iterations so that `x` will only ever refer to the value 1. With this, we have, $i_0 = n-1$, $i_1 = n-2$, $i_2 = n-3$, $\ldots$, $i_k = n-(k+1)$. Since i is initialized to the last index in the list and iterates in reverse, we know that the loop terminates when $i_k \leqslant 0$. Then we want to find the smallest value of k such that $i_k \leqslant 0$. So we have,

$$\begin{aligned} i_k &\leqslant 0 \\ n-(k+1) &\leqslant 0 \\ n-1 &\leqslant k \end{aligned}$$

and we see that the loop will terminate when $k = n-1$. Thus there are *at most* $n-1$ steps, so $WC_{myprogram}(n) \in \mathcal{O}(n)$.

□

(b) Prove $WC(n) \in \Omega(n)$.

Proof. Lower bound on the worst-case:

We want to choose an input family such that for any length n, there exists a list `L` such that the runtime of `myprogram` runs in *at least* linear time.

Let `n = len(L)` be the length of the input list to `L`. For any list of length n, let `L` be the list of all 1's. That is, let

```
L[i] = 1
```

for all indices i in `range(len(L))`. Then `line 6` always evaluates to `False`, and we enter the `else` block on each iteration. Since we never changed the value of `x`, i always decreases by 1 and the analysis is similar to the worst-case. Since the loop terminates when $i_k \leqslant 0$, we want to find the *largest* value of k such that $i_k > 0$ (i.e. the loop continues). From our previous analysis for the upper bound on the worse-case, we found that $i_k = n - (k+1)$. Then,

$$\begin{aligned} i_k &> 0 \\ n - (k+1) &> 0 \\ n - 1 &> k \end{aligned}$$

So we choose $k = n - 1$. This shows that there are *at least* $n - 1$ iterations in the worst-case, with each iteration costing a single step. Then $WC_{myprogram}(n) \in \Omega(n)$.

□

(c) Prove that $BC(n) \in \mathcal{O}(\log n)$.

Proof. Upper bound on the best-case:

We want to show that for any list of length n, there exists an input family of lists `L` such that the runtime of `myprogram` grows *at most* logarithmically.

Let `n = len(L)` be the length of the input list, `L`. Then for any list of length n, let `L` be the list of all 2's. That is, let

```
L[i] = 2
```

for all indices i in `range(len(L))`. Then `line 6` always evaluates to `True`, and we always enter the `if` block, so i decreases by a factor of 2 after each iteration. Then we have, $i_0 = \left\lfloor \frac{n-1}{1} \right\rfloor$, $i_1 = \left\lfloor \frac{n-1}{2} \right\rfloor$, $i_2 = \left\lfloor \frac{n-1}{4} \right\rfloor$, $\ldots$, $i_k = \left\lfloor \frac{n-1}{2^k} \right\rfloor$. We know the loop terminates when $i_k \leqslant 0$, so we want to find the smallest value of k such that $i_k \leqslant 0$. This happens one step after $i_k = 1$. But i is initialized to $n-1$, so we have,

$$\begin{aligned} 1 &\geqslant i_k \\ 1 &\geqslant \frac{n-1}{2^k} \\ 2^k &\geqslant n-1 \\ k &\geqslant \log_2(n-1) \end{aligned}$$

So we choose $k = \lceil \log_2(n-1) \rceil$. This shows that the loop iterates *at most* $k = \lceil \log_2(n-1) \rceil$ steps in the best-case. By logarithmic Big-Oh properties, $\lceil \log_2(n-1) \rceil \in \mathcal{O}(\log n)$. Therefore, the total cost for the best-case is $\mathcal{O}(\log n)$ and we can conclude that $BC_{myprogram}(n) \in \mathcal{O}(\log n)$.

□

(d) Prove that $BC(n) \in \Omega(\log n)$.

Proof. Lower bound on the best-case:

We want to show that for any arbitrary list `L` of length n, the runtime of `myprogram` grows *at least* logarithmically.

Let `n = len(L)` be the length of the input list `L`. Also, let $k \in \mathbb{N}$ and let i_k be the value of the index i after k loop iterations.

The loop terminates when $i_k \leqslant 0$. There are two ways in which the loop counter can decrement on a given loop iteration, depending on the truth value of `line 6` in the code. The best case occurs when the loop counter decrements the most on each iteration, since this causes the least iterations of the loop. So in this case, we want a lower bound on the loop counter i.

Showing that `i -= x` cannot decrement the loop counter faster than dividing by 2:

The largest value that `x` can possibly be is $\lceil \log_2 n \rceil$. Now, let us assume that $x = \lceil \log_2 n \rceil$ on every iteration. Note that if we can prove that dividing by two is a quicker runtime than subtracting by $\lceil \log_2 n \rceil$, then it will be quicker to always run the `if`-statement.

The runtime of dividing by two is $\mathcal{O}(\log_2 n)$, and the runtime of subtracting by $\lceil \log_2 n \rceil$ is $\mathcal{O}(\frac{n}{\log_2 n})$. We will now show $\log_2 n \in \mathcal{O}(\frac{n}{\log_2 n})$:

$$\exists c_0, n_0 \in \mathbb{R}^{\geq 0} st. \forall n \in \mathbb{N}, n \geq n_0 \implies \log_2 n \leq c_0 \cdot \frac{n}{\log_2 n}$$

By Big-Oh hierarchy theorem we know that $\log_2 n \in \mathcal{O}(n^{\frac{1}{2}})$:

(1) $\exists c_1, n_1 \in \mathbb{R}^{\geq 0} st. \forall n \in \mathbb{N}, n \geq n_1 \implies \log_2 n \leq c_1 \cdot n^{\frac{1}{2}}$

Let $c_1, n_1 \in \mathbb{R}^{\geq 0}$such that the above holds.

Let $c_0 = (c_1)^2$, and let $n_0 = max(n_1, 1)$

Let $n \in \mathbb{N}$.

Assume $n \geq n_0$, then it follows that $n \geq n_1$ and $n \geq 1$.

From (1), since $n \geq n_1$:

$$\log_2 n \leq c_1 \cdot n^{\frac{1}{2}}$$

(Since both sides are positive) $$(\log_2 n)^2 \leq (c_1)^2 \cdot n$$

$$\log_2 n \leq c_0 \cdot \frac{n}{\log_2 n}$$

Since $\log_2 n \in \mathcal{O}(\frac{n}{\log_2 n})$, $\log_2 n$ has a faster runtime than $\frac{n}{\log_2 n}$. Thus, dividing by two will be faster than subtracting by $\lceil \log_2 n \rceil$.

The largest change for a given iteration of the loop is when i decreases by a factor of 2, and this occurs when the `if` block executes.

We want to find the smallest value of k such that the loop terminates, which occurs one step after $i_k \leq 1$. So we have,

$$\begin{aligned} 1 \geqslant i_k &= \frac{n-1}{2^k} \\ 2^k &\geqslant n-1 \\ k &\geqslant \log_2(n-1) \end{aligned}$$

This shows that the loop iterates *at least* $\lfloor \log_2(n-1) \rfloor + 1$ times. By logarithmic Big-Oh properties, $\lfloor \log_2(n-1) \rfloor \in \Omega(\log_2 n)$. Therefore, the total cost for the best case is $\Omega(\log n)$ and we can conclude that $BC(n) \in \Omega(\log n)$.

$\square$

3. Graph algorithm

(a) Prove that $WC(n) \in \Theta(2^n)$.

Proof. To prove a tight bound, we must prove both an upper bound and a lower bound.

Upper bound on the worst-case:

We want to show that for any arbitrary graph $G = (V, E)$ with $|V| = n$, the runtime of `has_isolated` is at most like 2^n.

Let G = (V, E) $\in G^{'}$ be an arbitrary graph [1]. Let M be the corresponding adjacency matrix of G. Let `n =` $|V|$ be the number of vertices in G. For a fixed iteration of the outer loop, the inner loop iterates `at most` n times, with each iteration costing a single basic step. The outer loop iterates `at most` n times as well, so to calculate the cost of the outer loop, we multiply by the cost of the inner loop. Thus we get

$$n \cdot n = n^2$$

steps (ignoring constant terms).

The third loop (on `line 14` in the code) iterates *at most* 2^n times, with each iteration costing a basic step. Then to get the total cost, we must add the cost for each block of code. So we get $n^2 + 2^n$. But since $n^2 \in \mathcal{O}(2^n)$ (by Big-Oh Sum Theorem), we conclude that the $WC_{has\_isolated}(n) \in \mathcal{O}(2^n)$.

Lower bound on the worst-case:

We want to choose an input family such that for any $|V|$, there exists a graph G such that the runtime of `is_isolated` is *at least* like 2^n.

Let G be a graph with $|V| = n$ and $E = \{\}$. Let M be the corresponding adjacency matrix for G. Let $n = |V|$. Since $E = \{\}$, the corresponding adjacency matrix will only contain 0's as demonstrated below:

$$M[i][j] = 0$$

for i, j in `range(n)`. Then in this case `count` $= 0$, and so `found_isolated` always evaluates to `True` and loop 3 (on `line 14`) iterates 2^n times as a result. Thus, this shows that the total runtime is at least 2^n steps, and so the $WC_{has\_isolated}(n) \in \Omega(2^n)$.

Since we have shown matching upper and lower bounds on the worst-case, we can conclude that the $WC_{has\_isolated}(n) \in \Theta(2^n)$.

$\square$

[1]We use $G^{'}$ as the set of all graphs

(b) Prove that $BC(n) \in \Theta(n^2)$

Proof. To prove a tight bound, we must prove both an upper bound and a lower bound.

Upper bound on the best-case:

We want to show that for any graph G = (V, E) with $|V| = n$, there exists an input family of graphs G such that the runtime of `is_isolated` grows *at most* like n^2.

Let $G = (V, E)$ be a graph such that $\forall v_1, v_2 \in V, v_1 \neq v_2 \implies (v_1, v_2) \in E$. Then G is fully connected. Then the corresponding input matrix M that refers to the graph will contain all 1's except along the main diagonal, which will be 0's. Let $n = |V|$. So,

$$M[i][j] = \begin{cases} 1 & \text{if } i \neq j \\ 0 & \text{otherwise} \end{cases}$$

$\forall i, j$ in range(n). For a fixed iteration of the outer loop, the inner loop iterates n times, with each iteration a single step. Moreover, `count` never refers to 0 at the end of the inner loop, so `has_isolated` never refers to `True`. The outer loop iterates n times. Since the loops iterate independent of each other, we simply multiply the cost of the inner loop by the cost of the outer loop to get the total cost of this block of code. So, the total cost in this case is $n \cdot n = n^2$ steps (ignoring constant terms). Loop 3 on `line 14` in the code never runs because `has_isolated` is `False`. So, we can conclude that $BC_{has\_isolated}(n) \in \mathcal{O}(n^2)$.

Lower bound on the best-case:

We want to show that for any arbitrary graph G = (V, E) with $n = |V|$, the runtime of `is_isolated` grows *at least* like n^2.

Let $G = (V, E) \in G'$ be an arbitrary graph. Let M be the adjacency matrix for the graph G. Let $n = |V|$.

Case 1: `found_isolated` becomes `True`

For a fixed iteration of the outer loop, the inner loop iterates n times, with each iteration a single step. The outer loop iterates *at least* one time, but in this case, `found_isolated` is forced to refer to the boolean value `True` (otherwise the outer loop would iterate more than its minimum number of iterations). Then loop 3 on `line 14` in the code iterates 2^n steps as we have seen, so the total cost is `at least` $1 \cdot n + 2^n$ steps, which we know to be in $\Omega(n^2)$ (By Big-Oh hierarchy Theorem).

Case 2: `found_isolated` stays `False`

For a fixed iteration of the outer loop, the inner loop iterates n times. The outer loop also iterates n times, since no early breaks occur. So the cost of loop 1 is the number of iterations of the outer loop multiplied by the cost of the inner loop, which is also n. The cost in this case is n^2 steps. Loop 3 iterates *at least* zero times. Then in this case the total cost is $n^2 + 0$ steps.

In either case, we can conclude that the runtime is *at least* n^2 steps. Thus, we have shown that $BC_{has\_isolated}(n) \in \Omega(n^2)$.

Since we have shown that the $BC_{has\_isolated}(n) \in \mathcal{O}(n^2)$ and $BC_{has\_isolated}(n) \in \Omega(n^2)$, we can conclude that $BC_{has\_isolated}(n) \in \Theta(n^2)$.

$\square$

(c) Let $\zeta(n)$ represent the number of adjacency matrices of size n-by-n that represent valid graphs. Then we have

$$\zeta(n) = 2^{\frac{(n-1)(n)}{2}}$$

(d) Prove that the number of adjacency matrices of size n-by-n that represent valid graphs is $2^{\frac{(n-1)(n)}{2}}$.

Proof. Let $G = (V, E) \in G'$ be an arbitrary graph. Let $n = |V|$. Let M be the n-by-n adjacency matrix as given in question 3. Let $\zeta(n)$ represent the number of adjacency matrices of size n-by-n that represent valid graphs. We want to show that $\zeta(n) = 2^{\frac{(n-1)(n)}{2}}$.

Since the adjacency matrix is n-by-n, there are n^2 possible entries in the matrix. But for all adjacency matrices, the main diagonal is always zeros (since entry M_{ij} cannot be adjacent to itself. i.e. there are no loops. This is given by the way in which we defined our graphs). Thus there are n zeros along the diagonal of any n-by-n adjacency matrix. But we also know that our adjacency matrices are symmetric, so there are only half as many possible entries. Counting the number of remaining positions, we get

$$\frac{n^2 - n}{2} = \frac{n(n-1)}{2}$$

possible positions. Another way to see this is by summing the number of possible variations, which is

$$\begin{aligned}(n-1) + (n-2) + \cdots + 1 &= \sum_{i=1}^{n-1} i \\ &= \frac{(n-1)n}{2}\end{aligned}$$

The values at each of these positions can be either 0 or 1, so the number adjacency matrices of size n-by-n that represent valid graphs is $2^{\frac{(n-1)(n)}{2}}$. As a check, this result makes sense because the positions of an adjacency matrix represent edges, and we know that the maximum number of possible edges of a graph is $\frac{(n-1)(n)}{2}$, where $|V| = n$ is the number of vertices. Then, since each position can be either 0 or 1, we raise our result to the power of 2 to obtain the number of possible n-by-n adjacency matrices. Therefore,

$$\zeta(n) = 2^{\frac{(n-1)(n)}{2}}$$

$\square$

(e) Let $n \in \mathbb{N}$. Prove that the number of n-by-n adjacency matrices that represent a graph with at least one isolated vertex is at most $n \cdot 2^{(n-2)(n-1)/2}$.

Let $Adj(n)$ be the number of n-by-n adjacency matrices that represent a graph with at least one isolated vertex. We want to prove that,

$\forall n \in \mathbb{N}, \forall G = (V, E), G$ is not connected $\implies Adj(n) \leqslant n \cdot 2^{(n-2)(n-1)/2}$.

In other words, we get to assume here that G is not connected since it has at least one isolated vertex.

Proof. Let $n \in \mathbb{N}$ and let $G = (V, E)$ be an arbitrary graph. Assume that G is not connected. Let $k \in \mathbb{N}$ and assume v_k is the isolated vertex. This means that there is no such path of vertices $\{v_1, \ldots, v_n\} \in V$ to v_k. In terms of the adjacency matrix, this means that there must be *at least* one row and column of all zeros. Since the entries in an adjacency matrix represent the number of possible edges from a graph, we first want to find the number of possible edges from a graph with *at least* one isolated vertex. If we take the same approach to finding the number of possible positions in the adjacency matrix as we did in question $3(d)$, then we notice that there are *at most* $n - 1$ less positions than the maximum number positions, which we found in question $3(d)$. This is because there is *at least* one row and column of all zeros. If we start with the number of possible entries in the adjacency matrix that we found in question $3(d)$ and then remove $n - 1$ entries, we get the number of possible remaining entries

$$\begin{aligned} \frac{n^2 - n}{2} - (n-1) &= \frac{n(n-1)}{2} - (n-1) \\ &= \frac{n(n-1) - 2(n-1)}{2} \\ &= \frac{(n-2)(n-1)}{2} \end{aligned}$$

Once again, since each entry in the adjacency matrix can be either a 1 or a 0, we need to raise this result to the power of 2. But we aren't finished yet because we also need to account for the isolated vertex v_k being any one of the n vertices of the graph. In other words, the isolated vertex can be any of $\{v_1, \ldots, v_n\} \in V$. In terms of the adjacency matrix, this means that the row/column of zeros can be in any of the n rows/columns, so we need to multiply our final result by n. Putting this all together, we get

$$\begin{aligned} Adj(n) &\leqslant n \cdot 2^{(n-2)(n-1)/2} \\ &= n \cdot \zeta(n-1) && \text{(As defined in Q3 d)} \end{aligned}$$

□

(f) Let $AC(n)$ be the average-case runtime. Prove that $AC(n) \in \Theta(n^2)$.

Proof. Let $G = (V, E)$ be an arbitrary graph that represents the input matrix `M`. Also, let

$$Ave(n) = \frac{\text{sum of all runtimes for all valid adjacency matrices}}{\text{number of valid inputs}}$$

But we know that the number of valid inputs is simply the value that we calculated from question $3(c)$, which we defined as $\zeta(n) = 2^{n(n-1)/2}$. So that takes care of the denominator. For the numerator, lets define S_n as the sum of all runtimes for all valid adjacency matrices (i.e. the numerator). But we notice that we can split S_n into the sum of runtimes with no isolated vertex, and the sum of runtimes with at least one isolated vertex. That is,

$$\begin{aligned} S_n = &\text{ sum of RTs with no isolated vertex} \\ &+ \text{sum of RTs with at least one isolated vertex} \end{aligned}$$

We also know that the number of n-by-n adjacency matrices with at least one isolated vertex is at most $n \cdot 2^{(n-1)(n-2)/2}$. With this, and also knowing total number of valid n-by-n adjacency matrices (Q3(c)), we can find the number of n-by-n adjacency matrices with no isolated vertex. That is, let $\zeta(n)$ be the total number of valid inputs as we defined above and in question $3(c)$, let $Iso(n)$ be the number of inputs with at least one isolated vertex, and let $\neg Iso(n)$ be the number of inputs with no isolated vertex. Then we can write,

$$\begin{aligned} \neg Iso(n) &= \zeta(n) - Iso(n) \\ &= 2^{n(n-1)/2} - n \cdot 2^{(n-1)(n-2)/2} \end{aligned}$$

Next, we want to calculate S_n now knowing $Iso(n)$ and $\neg Iso(n)$. We also know that an arbitrary graph with no isolated vertex and it's corresponding adjacency matrix will always have a runtime of n^2, since the first two loops run fully, and the third loop will never run. We now need to find the runtime of an adjacency matrix coresponding to a graph with at least one isolated vertex. By looking at the code, we see that the loop on line 14 must run, (since this group of inputs is assumed to have *at least* one isolated vertex. The runtime of the double nested loops requires more attention.

Let Ψ be the first occurrence of a row of zeros in M. Note that $0 \leq \Psi \leq n$. In this case, Ψ, refers to the first isolated vertex.

Notice that the loop counter for loop 5, will break once $i = \Psi$. For any fixed iteration of the outer loop, the inner loop still iterates n times. So the total runtime the sum of both blocks of code:

$$n \cdot \Psi + 2^n$$

So then to calculate S_n we need to multiply these runtimes by the number of possible adjacency matrices. That is,

$$\begin{aligned}
S_n &= n^2\big[\neg Iso(n)\big] + (n\cdot\Psi + 2^n)\big[Iso(n)\big] \\
&= n^2\Big[2^{n(n-1)/2} - n\cdot 2^{(n-1)(n-2)/2}\Big] + (n\cdot\Psi + 2^n)\Big[n\cdot 2^{(n-1)(n-2)/2}\Big] \\
&= n^2\Big[2^{n(n-1)/2}\Big] - n^2\Big[n\cdot 2^{(n-1)(n-2)/2}\Big] + \\
&\quad + n\cdot\Psi\Big[n\cdot 2^{(n-1)(n-2)/2}\Big] + 2^n\Big[n\cdot 2^{(n-1)(n-2)/2}\Big]
\end{aligned}$$

Now that we have calculated S_n, we can go back to our original problem of solving $Ave(n)$. Thus, we have

$$\begin{aligned}
Ave(n) &= \frac{S_n}{\zeta(n)} \\
&= \frac{n^2\Big[2^{n(n-1)/2}\Big] - n^2\Big[n\cdot 2^{(n-1)(n-2)/2}\Big] + n\cdot\Psi\Big[n\cdot 2^{(n-1)(n-2)/2}\Big] + 2^n\Big[n\cdot 2^{(n-1)(n-2)/2}\Big]}{2^{n(n-1)/2}} \\
&= n^2 - n^3\cdot 2^{(1-n)} + n^2\cdot\Psi\cdot 2^{(1-n)} + 2n
\end{aligned}$$

Note, $\frac{n^a}{2^n} \in \mathcal{O}(1)$, where $a \in \mathbb{R}^{\geq 0}$, by an extension of the Big-Oh Hierarchy Theorem. So, asymptotically, the terms $n^3\cdot 2^{(1-n)} + n^2\cdot\Psi\cdot 2^{(1-n)}$ will not affect the theta bound. So,

$$Ave(n) = n^2 + 2n$$

Thus, we see that the average runtime is $n^2 + 2n$. Since $n^2 + 2n \in \Theta(n^2)$ by the Big-Oh Hierarchy Theorem, we conclude that $AC_{has\_isolated}(n) \in \Theta(n^2)$.

$\square$