CSC165H1 Problem Set 4

Yulin WANG, Qidi Zhou, Dana Zhao

Wednesday April 4

1. Binary representation and algorithm analysis

a)

Proof:

Let n $\in \mathbb{N}$, consider the running time of count on an input of n.
Let's consider the loop from line 9 to line 16 first.
The outer loop has n iterations, and for each iteration, the inner while loop
from line 13 to line 15 takes at most $\lfloor \log_2 n \rfloor$ steps,
since the cost of while loop is depend on j that is $\lfloor \log_2 n \rfloor$.
And since the inner loop takes constant time for each iteration,
so the total cost of the loop from line 9 to line 16 is n($\lfloor \log_2 n \rfloor$).
Since code on line 6 and line 7 takes constant steps, we simply count constant steps as 1,
so the total running time of function count is at most 1 + n($\lfloor \log_2 n \rfloor$), which is $\mathcal{O}(nlogn)$.

□

b)

Proof:

Let n $\in \mathbb{N}$, consider the running time of count on an input of n.
Let's consider the loop from line 9 to line 16 first.
Let's consider inner while loop first. The inner while loop from line 13 to line 15 takes
at least one step. And since the outer loop has n iterations,
with taking at least constant time for each iteration,
so the total cost of for loop takes n steps.
Let's consider line 6 and line 7 which takes constant time,
so the total running time of function count is at least n + 1 ,which is Ω(n).

□

2. Worst-case and best-case algorithm analysis

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

Proof:

Let $n \in \mathbb{N}$, and consider the running time of function myprogram on an input of length n.

First, let's consider the while loop from line 5 to line 10.
Since the value of i decreases at least one after each iteration,
so the while loop has at most $(n-1)$ iterations (for $i = n-1, ..., 1$)
And for each iteration, it takes constant time,
thus the cost of the loop is at most $(n-1)$ steps.
Since code from line 2 to line 4 takes constant time,
which can be counted as one single step,
so the total running time of function myprogram is at most $(n-1)+1 = n \in \mathcal{O}(n)$
Hence, $WC(n) \in \mathcal{O}(n)$

□

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

Proof:

We need to find an input family for function myprogram whose running time is $\Omega(n)$.
For each $n \in \mathbb{N}$, consider the list L where L[i] = 1 for all i $\in$ {1, 2, ..., n-2, n-1}
In this case, the condition on line 6 is always False for each iteration,
so the value of i decreases by one after each iteration.
Thus the while loop from line 5 to line 10 has $(n-1)$ iterations.
And since each iteration takes constant time, the total cost of the loop is $(n-1)$ steps.
Since the code from line 2 to line 4 takes constant time,
so the total running time of function myprogram is $(n-1)+1 = n \in \Omega(n)$
Thus, $WC(n) \in \Omega(n)$

□

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

Proof:

We need to find an input family for function myprogram, whose running time is $\mathcal{O}(\log n)$
For each $n \in \mathbb{N}$, consider the list $L[i] = 2$ for all i $\in$ {1, 2, ..., n-2, n-1}
In this case, the condition on line 6 is always True for each iteration,
so the value of i changes by code $i = i//2$ on line 7.
So the loop from line 5 to line 10 has at most $\log_2(n-1)$ iterations,
and it takes constant time for a fixed iteration.
Thus, the total cost of the loop is $\log_2(n-1)$ steps.
Since the code from line 2 to line 4 takes constant time,
so the total running time of function myprogram is $\log_2(n-1) + 1 \in \mathcal{O}(\log n)$
Thus $BC(n) \in \mathcal{O}(\log n)$

□

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

Proof:

First, we claim that if there is one even case and one odd case in the while loop when the variable isn't too big, the best case must be running even case before running odd case, since the value of i decreases faster when $L[i]$ is even.
Assume that the even case takes k steps in total.
Since from (c) we know that the loop has at most $\log_2(n-1)$ iterations
when elements in list L are all even numbers,
so we can let $k = m \log n$, where $0 < m < 1$.
Assume the odd case takes p steps.
Then we have:
$p \geq \frac{\frac{n}{2^k}}{k}$ (where $\frac{n}{2^k}$ is the value of i after running all the even cases)
$= \frac{n}{2^k \cdot k} = \frac{n}{2^{m \log n} \cdot m \log n} = \frac{n}{n^m \cdot m \log n}$
$= \frac{n^{1-m}}{m \log n} \in \Omega(\log n)$ (since $0 < 1 - m < 1$ and $n^{1-m} >> \log n$)
Thus, $BC(n) \in \Omega(\log n)$

□

3. Graph algorithm

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

Proof:

Step1: Prove the upper bound: WC(n) $\in \mathcal{O}(2^n)$

Let n $\in \mathbb{N}$, and consider the running time of has_isolated on an input of length n.
Let's consider the loop from line 5 to line 11 first.
The outer loop has at most n iterations, and for each iteration, the inner loop has at most n iterations, with each iteration taking a single step(constant time bock of code).
Thus, the total cost of the loop from line 5 to line 11 is at most n^2 steps.
The loop on line 14 and 15 has at most 2^n iterations, with each iteration taking a single step, so the cost of this loop is 2^n steps.
And since the code on line 2 and 3 takes constant time, so the total running time of function has_isolated is at most $(1+n^2+2^n) \in \mathcal{O}(2^n)$.
Thus, WC(n) $\in \mathcal{O}(2^n)$.

Step2: Prove the lower bound: WC(n) $\in \Omega(2^n)$

We need to find an input family for function has_isolated, whose running time is $\Omega(2^n)$.
For each n $\in \mathbb{N}$, consider M[0][j] = 0 for all j $\in \{0,1,...,n-2,n-1\}$,
which means there is an isolated vertex.
In this case, in the first iteration of the outer loop on line 5,
the inner loop on line 7 has n iterations, with each iteration taking one single step.
And after n iterations of the inner loop, the value of count is 0,
so the condition on line 9 is True, and found_isolated = True, then break the loop.
Thus, the total cost of the loop from line 5 to line 11 is n steps.
Since found_isolated is True, so the condition on line 13 is True,
then the loop on line 14 has 2^n iterations, with each iteration taking one single step,
so the cost of this loop is 2^n steps.
And since the code on line 2 and 3 takes constant time,
so the total running time of function has_isolated is $(1+n+2^n) \in \Omega(2^n)$.
Thus, WC(n) $\in \Omega(2^n)$.

In conclusion of step1 and step2, we've proven that the worst-case running time of this algorithm is $\Theta(2^n)$.

□

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

Proof:

Step1: Prove the upper bound: BC(n) $\in \Omega(n^2)$.

Let n $\in \mathbb{N}$, consider the running time of function has_isolated on an input of length n.
Let's consider the loop from line 5 to line 11 first.
The outer loop on line 5 has n iterations, and for each iteration,
the inner loop has n iterations, with each iteration taking one single step.
Then the cost of the whole loop is n^2 steps.
When the value of count is not 0, then the condition on line 13 is False,
so the for loop on line 14 doesn't run any time.
And since code on other lines takes constant time.
So, the total running time of function has_isolated is at least $(n^2 + 1) \in \Omega(n^2)$.
Thus, BC(n) $\in \Omega(n^2)$

Step2: Prove the upper bound: BC(n) $\in \mathcal{O}(n^2)$.

We need to find an input family for function has_isolated, whose running time is $\mathcal{O}(n^2)$.
For each n $\in \mathbb{N}$, consider M[i][j] = 1 for all i,j $\in$ {0,1,...,n-2,n-1}.
In this case, the value of count doesn't become 0 through n^2 iterations.
So the condition on line 13 is always False,
then the for loop on line 14 doesn't run any time.
And since code on other lines take constant time.
So the total running time is $(1 + n^2) \in \mathcal{O}(n^2)$.
Thus, BC(n) $\in \mathcal{O}(n^2)$.

In conclusion of step1 and step2, we've proven that the best-case running time of this algorithm is $\Theta(n^2)$.

□

c)

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

d)

Prove: $\forall$ n$\in \mathbb{N}$, the number of adjacency matrices of size n-by-n that represent valid graphs is:

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

Proof by induction:

Let P(n) be the predicate "the number of adjacency matrices of size n-by-n that represent valid graphs is $2^{\frac{n(n-1)}{2}}$.", where n is a natural number.

We want to prove that $\forall$ n$\in \mathbb{N}$,P(n).

Base case: Let n = 0.

In this case, since the number of adjacency matrices of size 0-by-0 that represent valid graphs is:

$$1 = 2^{\frac{0(0-1)}{2}} = 2^0.$$

So P(0) is True.

Induction step:

Let n $\in \mathbb{N}$. Assume that P(n) is True, that is the number of adjacency matrices of size n-by-n that represent valid graphs is

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

Want to prove that P(n+1) is True,

that is he number of adjacency matrices of size (n+1)-by-(n+1) that represent valid graphs is

$$2^{\frac{n(n+1)}{2}}.$$

Consider the first n vertices,

we know that there are $2^{\frac{n(n-1)}{2}}$ adjacency matrices by our induction hypothesis.

And for the (n+1)th vertex, there exist 2 possibilities for each of other n vertices,

so there are exactly 2^n possibilities.

Thus, combining the first n vertices and the (n+1)th vertex,

there are $2^{\frac{n(n-1)}{2}} \cdot 2^n = 2^{\frac{n(n-1)}{2}+n} = 2^{\frac{n(n+1)}{2}}$ adjacency matrices.

Thus, P(n+1) is True.

Therefore, we've proven that $\forall$ n$\in \mathbb{N}$, the number of adjacency matrices of size n-by-n that represent valid graphs is $2^{\frac{n(n-1)}{2}}$.

□

e)

Proof:

According to 3(c) and 3(d), we have:
If there are n vertices, then we have $2^{\frac{n(n-1)}{2}}$ possible adjacency matrices.
Since we want to know how many adjacency matrices at most there are with at least one isolated vertex, so we should consider the situation where we have just one isolated vertex.
In this case, for the other (n-1) non-isolated vertices,
there are $2^{\frac{(n-1)(n-2)}{2}}$ adjacency matrices. (by question 3(c))
And since we have n choices to determine which vertex to be the isolated one,
so there are $n \cdot 2^{\frac{(n-1)(n-2)}{2}}$ adjacency matrices in total.

Therefore, we've proven that the number of adjacency matrices of size n-by-n that represent a graph with at least one isolated vertex is at most $n \cdot 2^{\frac{(n-1)(n-2)}{2}}$.

□

f)

Proof: From question 3(c), we know that $\mathcal{I}_n = 2^{\frac{n(n-1)}{2}}$

Step1: Prove the lower bound: AC(n) $\in \Omega(n^2)$

Since from question 3(b), we know that the best running time is $\Theta(n^2)$ for each case.

So we can know that the cost of each case is: $\geq cn^2$ times, where $c \in \mathbb{R}^+$.

And since there are $2^{\frac{n(n-1)}{2}}$ cases,

so AC(n) $\geq \frac{1}{2^{\frac{n(n-1)}{2}}} \cdot 2^{\frac{n(n-1)}{2}} \cdot cn^2 = cn^2 \in \Omega(n^2)$

Thus, AC(n) $\in \Omega(n^2)$.

Step2: Prove the upper bound: AC(n) $\in \mathcal{O}(n^2)$

There are two cases:

1)There is at least one isolated vertex. and 2)There is no isolated vertex.

From question 3(e), we know the number of adjacency matrices of size n-by-n that represent a graph with at least one isolated vertex is at most $n \cdot 2^{\frac{(n-1)(n-2)}{2}}$.

From question 3(a), we know that the worst running time is $\Theta(2^n)$ for each case.

Thus, we have:

$$
\begin{aligned}
AC(n) &\leq \frac{1}{2^{\frac{n(n-1)}{2}}} \cdot (n \cdot 2^{\frac{(n-1)(n-2)}{2}} \cdot c \cdot 2^n + |2^{\frac{n(n-1)}{2}}| \cdot c'n^2) \quad (where\ c, c' \in \mathbb{R}^+) \\
&= cn \cdot 2^{\frac{(n-1)(n-2)}{2} - \frac{n(n-1)}{2} + n} + c'n^2 \\
&= cn \cdot 2^{\frac{n^2-3n+2-n^2+n+2n}{2}} + c'n^2 \\
&= cn \cdot 2^{\frac{2}{2}} + c'n^2 \\
&= 2cn + c'n^2 \in \mathcal{O}(n^2)
\end{aligned}
$$

Therefore, AC(n) $\in \mathcal{O}(n^2)$

In conclusion of step1 and step2, we've proven that AC(n) $\in \Theta(n^2)$

□