HW5

Jiayao Wu jwu86

March 3rd 2016

1 Problem 1

From class, we know that the running time for division into $\frac{n}{5}$ is

$$T(n) = O(n) + T(\frac{n}{5}) + T(\frac{7n}{10})$$

which is $O(n)$ as proved in class.

Similarly, the running time for division into $\frac{n}{3}$ groups is

$$T_1(n) = O(n) + T(\frac{n}{3}) + T(\frac{2n}{3}).$$

This is because first, it costs $O(n)$ in step 2 and 4. Step 2 is finding the medians of each of the $\frac{n}{3}$ groups. Step 4 is partitioning the original array with median x. Second, step 3 is recursively finding the median x of the $\frac{n}{3}$ medians found in step 2, so it is $T(\frac{n}{3})$. Lastly, step 5 of recursion on the low side or high side is $T(\frac{2n}{3})$ because at least half of the medians found in step 2 are less than x, so it's $\frac{n}{6}$. Then, since for each group of 3, if y is the median, then there are at least two numbers $\leq y$, so it's $2*\frac{n}{6} = \frac{n}{3}$. Then, the worst case to do the recursion is $n - \frac{n}{3} = \frac{2n}{3}$.

With similar reasoning, the running time for division into $\frac{n}{7}$ groups is

$$T_2(n) = O(n) + T(\frac{n}{7}) + T(\frac{5n}{7}).$$

Now we need to compare two $T(n)$.

We have actually solved $T_1(n)$ in HW3, $T_1(n) = O(nlogn)$. Now let's look at $T_2(n)$.

In class, we proposed $T_2(n) \leq dn$ where d is $O(1)$. Let $d \geq 7c$. We need to use induction to prove it here:

Base case: when $n = 1$, $T_2(n) \leq c * 1 + 0 + 0$, then we just need to let $c \leq d$ to make $T_2(n) \leq dn$ valid.

Induction step: assume true for all $n' < n$, we need to prove it's also true

for n:

Then,

$$T_2(n') \leq c * n' + d(\frac{n'}{7}) + d(\frac{5n'}{7})$$

so

$$T_2(n) \leq c * n + d(\frac{n}{7}) + d(\frac{5n}{7}) = cn + \frac{6dn}{7} \leq \frac{6dn}{7} + \frac{dn}{7} = O(n).$$

Therefore, we can see that $T_2(n)$ is smaller and asymptotically faster because $O(n) < O(nlogn)$. It is the same as the running time for groups of 5. Therefore, the one with division into groups of 5 and 7 are asymptotically faster.

2 Problem 2

Algorithm overview:

First, we traverse through array A to find min and max.

Second, we assign $\frac{min+max}{2}$ as a target. Then we traverse the array A again and compare each $A[i]$ with the target where i is from 1 to n. Create a counter initialized to 0. Create variable L = -infinity and R = +infinity. If $A[i] <$ target , then increment counter. If also, $A[i] > L$, then replace L with $A[i]$. On the other hand, if $A[i] \geq$ target and $A[i] < R$, then replace R with $A[i]$.

After all the comparisons, if counter equals to $\frac{n}{2}$, just return L or R.

If not, we repeated do the recursion. If counter $< \frac{n}{2}$, then recurse on the upper part. Else, recurse on the lower part.

Here is the pseudo code:
For i = 1,2,3.....n
    Traverse through array A to find max and min
Algorithm: FindMedian(max, min, A)
int target = $\frac{min+max}{2}$; L = -infinity; R = inifinity; counter = 0
For i = 1,2,3.....n
    If $A[i] <$ target
        counter++
            If $A[i] > L$
                $L = A[i]$
     else
        If $A[i] < R$
            $R = A[i]$
End For
If counter = $\frac{n}{2}$
    Return R or L $- - - >$ we have found the median
ElseIF counter $< \frac{n}{2}$
    Return FindMedian(max, R, A)
 ElseIF counter $> \frac{n}{2}$
    Return FindMedian(L, min, A)
End of algorithm

Prove correctness:

The process of finding max and min is assumed to be correct because it's just simply traversing through the array.

Then in the for loop, we chose target to be $\frac{min+max}{2}$ because it gives us a number that is relatively close to the median. The counter keeps track of how many numbers in A there are that are smaller than target. L keeps track of the largest number that is smaller than $A[i]$ and R keeps track of the smallest number that is larger or equal to $A[i]$. This means that in the end, L and R are close to each other without any number in between them.

Outside the for loop, we know that in the case of a median, there would be $\frac{n}{2}$ numbers smaller than the median and $\frac{n}{2}$ larger than the median exactly.

Therefore, when counter equals to $\frac{n}{2}$, we know we have found the median, then just return L or R.

In the case of counter $< \frac{n}{2}$, we know that the L and R we found are too small for the median, so the median must be in the upper portion, so recursing on the upper portion of the array makes sense.

In the case of counter $> \frac{n}{2}$, we know that the L and R we found are too big for the median, so the median must be in the lower portion, so recursing on the lower portion of the array also makes sense.

Here is a proof of contradiction:

If the algorithm returns a number that is not the median, then this means either there are more than $\frac{n}{2}$ numbers that are larger than it, or there are more than $\frac{n}{2}$ numbers that are smaller than it. This is impossible because we only return a number if the counter is $\frac{n}{2}$ and obviously in this case, the counter is not $\frac{n}{2}$, then the number wouldn't be returned.

Running time analysis:

The first step: When traversing through the array A to find max and min, we know it's $O(n)$ because we are making n comparisons. Then we look at the execution of the for loop. It is also $O(n)$ because it executes n times. Then when doing the recursion, we know the recursion is idealy called $\log n$ times because ideally, we are essentially cutting the array into half each time if the max and min we found are ideal. For each recursion, the for loop is $O(n)$. Therefore, the total running time is $O(n \log n)$. As for the space complexity, since we only use a few variables here, it is constant space.

3 Problem 3

Use induction to prove. Suppose each city is a vertex and the high way between cities can be represented by a edge connecting two vertices.

First, let's consider some base cases (see the picture below). Suppose there are 3 vertices that are connected to each other. Since it's only one-way direction, then there are 8 cases. We can see that for each case, there exists a path from A to B such that all the 3 vertices are visited exactly once.

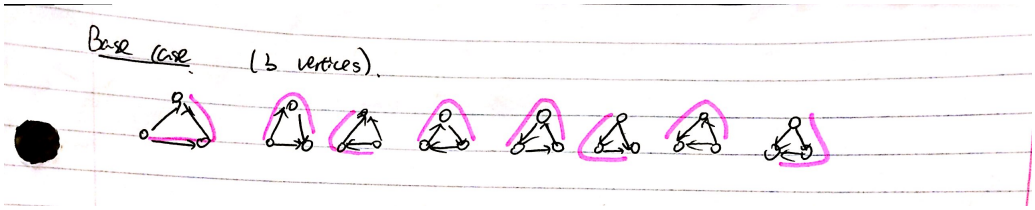

Figure 1: The Base Case

Second, suppose there exists a path from A to B in a connected, directed graph of n vertices such that each vertex is visited only once. Then, we need to prove that when adding a new vertex, meaning that it's now $n+1$ vertices, there still exists a path from A to B such that each vertex is visited only once (this A and B can differ from previous A and B). See the following figure 2 for 4 cases when a new vertex C is added. The dotted arrow from A to B represents the path of n vertices.

It is only showing one-way direction edge between A and C, B and C. All the other edges are just neglected here. There are 4 cases because each edge between A-C and B-C has two possible directions. For the first 3 cases, it's just changing either the start vertex (A) or the end vertex (B) to C in order to form the new path.

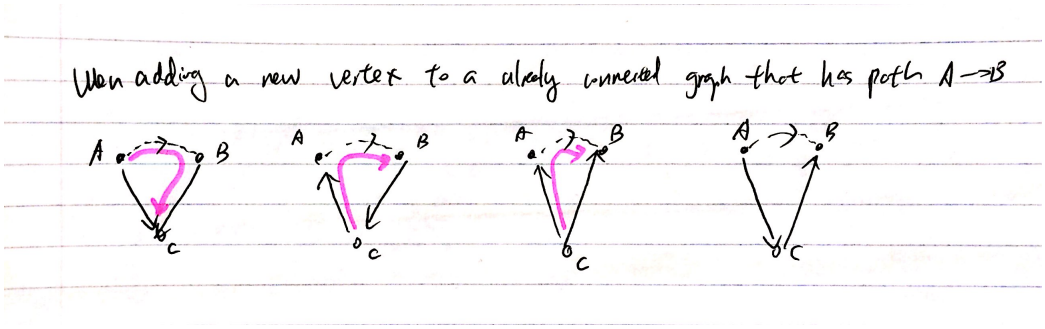

Figure 2: When adding a vertex to a connected, directed graph of n vertices

For example, for the first one, originally, the path started at A and end at B, but now after adding C, C becomes the new end vertex so that all the

vertices in the dotted line is visited as well. However, the last case needs more consideration. The following are 8 cases for it:

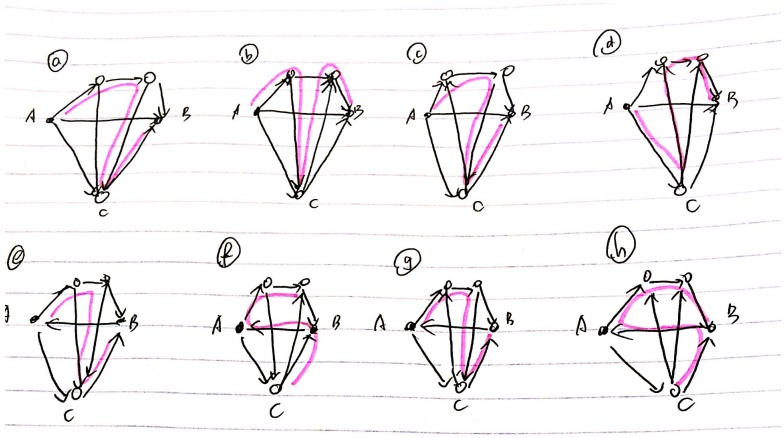

Figure 3: 8 cases

In this figure, we expanded the dotted line in figure 2 to 2 vertices and the edge between these 2 vertices may include many other vertices, called $\{s\}$, that are not shown here, so again A to B represent a path that visit n vertices. We need to make sure the new path formed upon adding the vertice C should also visit $\{s\}$, that's why the path of each case except b goes through $\{s\}$. The exception here is b because it is not going through $\{s\}$ directly, but rather it's using C as an intermediate step. The following is more explanation:

Proposition: We can always find two adjacent edges with opposite direction that connect two adjacent vertices, called E and F in $\{s\}$ with C so that there can be a path formed that covers all the vertices. See the following figure.

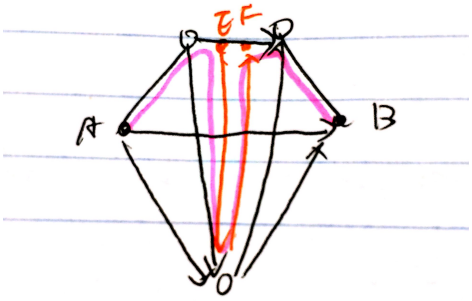

Figure 4: More specification for case b

We can prove this by induction. For the base case, if there is only one vertex in $\{s\}$, then both directions that connect with C will work because the left edge

and right edge of the middle triangle is of opposite direction.

Then suppose, for n vertices in $\{s\}$, there exists a pair of connecting vertices that are of opposite direction. Then it is obviously true for the case of $n+1$ since there is already a pair that is of opposite direction.

Therefore, my proposition is correct.

Therefore, with all the cases considered, we can conclude that we can find a path from A to B that vists $n+1$ vertices exactly once. Thus, the existence of such A, B and a path from A to B for any given configuration of the highway's one way directions is proved.