1 CSE 548 Fall 2018 - Analysis of Algorithm - Homework 1

Name: Rohit Bhal
Solar ID: 112073893
NetID email address: rbhal@cs.stonybrook.edu / rohit.bhal@.stonybrook.edu

2

2.1

Q1. Textbook [Kleinberg & Tardos] Chapter 2, page 69, problem #8

Answer (a): We have k = 2 jars.
Let us take a step size of n^x. Now, total number of steps before the jar breaks on the j^{th} rung = $\frac{n}{n^x}$.
After reaching the highest point it breaks, therefore we will increase our step by the size of 1 from the $j - 1^{th}$ rung.
This will add n^x to the total steps.
Therefore, f(n) = n^{1-x} + n^x
To minimize the number of steps, both the terms should be equal.
Therefore, 1-x = x.
x = $\frac{1}{2}$

Let us consider each step of $\sqrt{n}$, where n is the highest rung of the ladder.
One of the strategy is to drop the jar at the multiple of $\sqrt{n}$ times.
When the jar breaks at j^{th} rung.We know the answer lies between the $(j-1)^{th}$ & j^{th} rung.
Therefore, at most we will iterate $\sqrt{n}$ steps, incrementing the step by one from $(j-1)^{th}$ rung. If the $\sqrt{n}$ is not perfect, we take the ceiling $\sqrt{n}$ as the step size.

$$f(n) = O(\sqrt{n})$$

$$\lim_{n\to\infty} \frac{f(n)}{n} = \frac{\sqrt{n}}{n} = \frac{1}{\sqrt{n}} = 0 \tag{1}$$

(b) : Let us take the step size as $n^{\frac{k-1}{k}}$ where k the is number of jars given & n is the highest rung of the ladder. For k = 2, we can use the solution of (a). Now for k > 2, the number of steps to reach the highest rung is:

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

Now, to move from $(j)^{th}$ to $(j-1)^{th}$ rung, we will require $n^{\frac{1}{k}}$ steps. Therefore, the total number of steps required as a function of n can be written as :

$$f_k(n) = 2 * n^{\frac{1}{k}} \tag{3}$$

$$\lim_{n\to\infty} \frac{f_k(n)}{f_{k-1}(n)} = \frac{2 * n^{\frac{1}{k}}}{2 * n^{\frac{1}{k-1}}} = \frac{n^{k-1}}{n^k} = \frac{1}{n} = 0 \tag{4}$$

2.2

Q2. Textbook [Kleinberg & Tardos] Chapter 3, page 107, problem #4 .
Answer: We have a collection of n specimens & m judgements (either "same" or "different") for the pair that were declared not to be ambiguous.
Now, we need to find an algorithm with running time O(m + n) that determines whether the m judgments are consistent.
We can take specimens as vertices & judgements as edges of the graph. If the pair(i,j) has a judgement of "same", we will label both the vertices with the same value and the one with "different" judgement will have different value.
Now, we can have an adjaceny list for each vertices in the graph consisting of it's neighbour. For a vertice v,we iterate through each of it's neighbour. If the edge(v,u) label is "same", the value of vertice u will be equal to it's parent and vice-versa for the edge(v,u) having label "different". If any of these conditions don't match, then we can say that the m judgements are not consistent. We can use BSF algorithm to traverse all the vertices since it is undirected graph. We can assign "A" value to the starting edge.

Algorithm:
Traversal(s):
color[s] = A
put (s) in the **queue**
while(queue $\neq \phi$)
take v from the queue
if v is not marked

```
    mark v
for each edge (v,w):
if(edge(v,w) == 'S'){
    if(color[w] ≠ NULL && color[v] ≠ color[w) {
       return false;
    }
    else {
       color[w] = color[v];
    if w is not marked put w in the queue }
} else {
    if(color[w] ≠ NULL && color[v] == color[w) {
       return false;
    } else {
       color[w] = (color[v] == A) ? B ? A;
    if w is not marked put w in the queue
    }
}
ENDFOR
return true;
```

2.3

Q3. Textbook [Kleinberg & Tardos] Chapter 3, page 107, problem #11 .
Answer: We are given n computers & m triples (C_i, C_j, t_k) indicating the communication between C_i & C_j at time t_k.
The C_a computer is infected at time x. We need to find whether C_b computer is infected by time y.
Let us take an Array infected of size n+1.
infected[n] = 0, denotes that the n^{th} computer is not infected
infected[n] = 1, denotes that the n^{th} computer is infected
Also, the triples are given in sorted order of time.
We have the m triplets in a vector< pair< pair<int,int>, int> > M. Where the first pair denotes the two computer communicating with each other & the last integer denotes the time of communication.

Algorithm:
For each item (C_i, C_j, t_k) in M:
if(t_k > y) {
    break ;

}
if(($C_i == C_a$ || $C_j == C_a$) && $t_k >=$ x) {
infected[C_i] = 1; infected[C_j] = 1;
} else if(($C_i == 1$ || $C_j == 1$)) {
infected[C_i] = 1; infected[C_j] = 1;
}
ENDFOR

if(infected[C_b] == 1] {
return true;
}
return false;

2.4

Q4.List the following functions in increasing asymptotic order. Between each adjacent functions in your list, indicate whether they are asymptotically equivalent (f(n) $\in \theta$(g(n)), you may use the notation that f(n) = g(n)) or if one is strictly less than the other (f(n) $\in$ o(g(n)) and use the notation that f(n) $\leq$ g(n)).
Answer:
1. $5 * n^3$+log(n) = $\theta(n^3)$
2. $min\{n^2, 1045 * n\} = \theta$(n)
3. $\sum_{i=1}^{n} i^{77} = \theta(n^{78})$
4. $ln(4) \approx 1.38629436112$.
Therefore $n^{ln(4)} \approx \theta(n^{1.3862})$
5. $\lceil \frac{n^2}{45} \rceil = \theta(n^2)$
6. $\lfloor \frac{n^2}{45} \rfloor = \theta(n^2)$
7. $\frac{n^2}{45} = \theta(n^2)$
8. $\log \sqrt{n} = \frac{1}{2} * \log n$
$= \theta(\log n)$
9.$\sum_{i=1}^{n} \frac{1}{i} \approx \theta$(log(n))
10. $ln(n!) = \sum_{i=0}^{n-1}$ log(n-i) = n* $\sum_{i=0}^{n-1}$ log(1- $\frac{i}{n}$)
$\approx \theta$(n * $\log n$)

The order of the functions is as follow :
$2^n > 3^{n/2} > 2^{n/3} > 2^{log(n)} > \sum_{i=1}^{n} i^{77} > 5 * n^3$+log(n) > $\lceil \frac{n^2}{45} \rceil = \frac{n^2}{45} = \lfloor \frac{n^2}{45} \rfloor >$

$n^{ln(4)} > ln(n!) > min\{n^2, 1045 * n\} > ln(n) = \sum_{i=1}^{n} \frac{1}{i} = log(\sqrt{n}) > lg(n)^{\sqrt{lg(n)}}$
$> \sqrt{lg(n)} > lg(lg(n)) > \sum_{i=1}^{n} \frac{1}{i^2} > \sum_{i=1}^{n} \frac{i^2+5*i}{6*i^4+7}$

2.5

Q5.An Euler tour of a graph G is a closed walk through G that traverses every edge of G exactly once.
(a) Prove that a connected graph G has an Euler tour if and only if every vertex has even degree.
(b) Describe and analyze an algorithm to compute an Euler tour in a given graph, or correctly report that no such tour exists.

Answer:(a) Let us assume that there exists an Euler tour in the G = (V,E). Let the Euler tour be v_0, v_1, ..., v_k. Since, Euler tour travels every edge only once, therefore k = |E|. Also, Euler tour is a closed walk, therefore $v_0 = v_k$. Now, the degree of the vertice u is the number of times it appears in the walk multiplied by two. Let u = v_i for 0 < i < k, then both $v_i - 1$, v_i, v_i, $v_i + 1$ are edges incident to u. Therefore, the degree of u is even.
Therefore, each vertex in the graph has even degree.

Now, let us assume that in the Graph G = (V,E), every vertex has even degree. Let us consider a closed walk W = v_0, v_1, v_2, ... , v_k , k =|E|
Now W is the longest walk in G that traverses no edge more than once. W must traverse every edge incident to v_i for 0 < i < k otherwise the walk could be extended and W would not be the longest walk. Since W is a closed walk, therefore $v_0 = v_k$. Since, every vertex has even degree, we can enter the vertice via one edge & exit through the other. Thus, Euler tour exists in the graph.
Suppose W is not the Euler tour.But G is connected graph, therefore there exist a new vertice not in W that is incident to some vertex in W. We can call this edge u, v_i. Now, we can construct a longer Walk W' by including u in the walk.
W' = u v_i, $v_i + 1$, ...v_k v_k, v_1, ... v_i
Since u has even degree.
Therefore, a connected graph G has an Euler tour if and only if every vertex has even degree.
(b) Algorithm to find Eulerian Path in a given graph :
Give G as an adjacency matrix representation of the graph, Where G(i,j) == 1 if there is path from i to j & G(i,j) == 0, if there doesn't exist any path

from i to j

Algorithm:

```
EulerPath(G):
List Edges
For each row i in G:
    For each item in row i:
        num = 0
        if( item == 1)
            num++;
        add num in Edge List
oddedges = 0
s = 0;
for i in Edges:
    if( i%2 == 1) {
        oddedges++;
        startpoint = i;
}
if(oddedges > 2) {
    return "No Solution"
}
stack
path
current = startpoint
while(stack is not empty & current has neighbours ) {
    if(current not has no neighbour) {
        add current to path
        current = stack.pop()
        }
    else {
        for each item in graph[current]:
            if(item == 1) {
                add current to stack
                remove the edge from current to item
                remove the edge from item to current
                break
            }
}
for each item ∈ path:
```

print item + '->'

Description:

Process to Find the Path:
1. First find the degree of each vertex in the graph. 2. Count the number of vertices having odd degree in the graph. If the number of oddedges is greater than 2 then the Euler cycle doesn't exist. 3.Take an empty stack and an empty path. If the count of oddedges > 0, then take the vertice having odd degree as the starting point.
4.If all the vertices has even number of edges then start from any of them.Set the current variable = startpoint.
5.If the current vertex has no neighbours. Then, add the current node to the path and pop the vertex from stack and set it as current.
6. If there are neighbours then iterate through the neighbours. Add the current node to the stack and remove the edge from the current vertex to the neighbour and vice-versa.
5.Repeat process 3 and 4, until the stack is empty and current node has not any neighbour.
The runtime complexity of this algorithm is $\theta(E)$
.

2.6

Q6.Prove that any connected acyclic graph with n >= 2 vertices have at least two vertices with degree 1. Notice that you should not use any known properties of trees and your proof should follow from the definitions directly.

Answer:Let us assume there exist a acyclic graph having no vertices of degree = 1. Consider the longest path in the graph G where every vertex is visited only once.
G = (V,E)
P = v_0, v_1, ... v_k , k = |E|
Now since the graph is acyclic. Therefore $v_0 \neq v_k$ and also any vertex v_i for $0 < i < k$ cannot be incident to vertext v_0 and v_k otherwise a cycle will be formed. But v_0 and v_k doesn't has degree = 1. Therefore, there exists a vertex incident to both of these vertices. Hence, the path can be extended. Thus, contradiction is found and our assumption is incorrect.
Now, we can say same about the acyclic graph having one vertex with degree one as either of v_0 or v_k will be having more than 1 degree. Thus, again making

a contradiction.
Hence,any connected acyclic graph with n >= 2 vertices have at least two vertices with degree 1.