Assignment 1

Vivek Bansal (Id : 112044493)
CSE 548 : Analysis of Algorithms
Discussed with : Ayushi Srivastava

Due Date : September 16, 2018

Ans 1. Glass Jar Problem

a) We have K = 2 jars.

Strategy : Let's say we drop our jar from xth rung. We have to choose x in such a way that number of steps going above x and going below x should be same. There are 2 cases possible:
i) Jar breaks after dropping from xth rung. In this case highest safe rung is x.
ii) Jar doesn't breaks after dropping from xth rung. Then we will try next rung to be x + (x-1) = 2x - 1, if it still doesn't break then will try next rung to be (3x-2) and so on until we will cross the highest possible rung.

= x + (x-1) + (x-2) + (x-3) + ... + (x-(x-1)) <= n
= x^2 - (x-1)(x-2)/2 <= n
= x^2 + 3x -2 <= 2n
= x <= $2\sqrt{n}$ (approximate to get upper bound)

So we realized that highest rung is bound by time $2\sqrt{n}$, so f(n) = $2\sqrt{n}$.
Now, $\lim_{n\to\infty} f(n)/n$ = $2\sqrt{n}$ / n
= $2/\sqrt{n}$
= 0.

b) We have k > 2 jars.

f_1 = n
f_2 = $2\sqrt{n}$
f_3 = $3^{n/3}$
By Induction :
Base case : When k == 1 it will require n rungs as we have to check linearly from the top. So f_1 = n which is true.

Now we will prove for k jars : $f_k(n) = kn^{1/k}$.

For k-1 jars, we have $f_{k-1}(n) = (k-1)n^{1/(k-1)}$

$f_{k-1}(n) <= (k-1)n^{1/k}$

For the first jar we require $n^{1/k}$ drops so adding that value to $f_{k-1}(n)$ will give us:

$<= (k-1)n^{1/k} + n^{1/k}$

$<= kn^{1/k} \quad = f_k(n)$ which proves our induction step.

Now, $\lim_{n\to\infty} f_k(n)/f_{k-1}(n) \quad = kn^{1/k}/(k-1)n^{1/(k-1)}$

$= n^{1/k}/n^{1/(k-1)}$

$= 1/n^{1/(k(k-1))}$

$= 0$

Ans 2. Specimens-Judgements Problem

We will construct a graph G=(V,E) in which vertices consist of specimens and edges denote the judgements between them. If we have a judgement between 2 specimens we will construct an edge between them. We will use breadth first traversal starting with any node stating it as starting node and label it as A. If there is an edge between this node and its neighbor, we will check for their corresponding judgement. If judgement is same we will mark them with same label, else if it is different we will mark them with different labels. For judgements which are arbitrary we will mark them with A. After this labelling, we will iterate through all the judgements and our labelling in the graph. If there is any mismatch in judgement with our corresponding labelling then there is inconsistency, else the judgements are consistent.

Algorithm:

Take an empty queue q and a visited array visited[ ].
Start with any node and label it as 'A' and push it in queue.

```
while(!q.empty())
    pop the front node(u) from queue and push all its neighbors(v) into queue.
    visited[u] = 1
    for each edge(u,v) label the nodes.
        if (visited[v] == 0)
            If judgement(u,v) is same, mark node v with same label as u.
            If judgement(u,v) is different, mark node v with different label than u.
            push v into queue.
            visited[v] = 1.
```

After labelling the graph we will iterate through all judgements and check for each judge-

```
ment(u,v).
For all given judgements(u,v):
    check for their corresponding labels in the above labelled graph.
    if label(u,v) is consistent with judgement(u,v) then continue;
    else return "inconsistent".
If we have iterated through all judgements then return "consistent".
```

Ans 3. ***Infected Computer Problem***

We will create a graph G=(V,E) in which vertices are the computers and edges denote the communication time. We will find the infected computer (C_a) at time (t_a) in the graph by doing normal traversal and then apply breadth first traversal considering this node as the starting node. After marking this node(u) as visited, we will push all its adjacent vertices(v) in the queue for which edge(u,v) lies between (t_a) and (t_b) and the timestamp of node v is greater than or equal to its parent(node u). We will continue the breadth first traversal until we will find our target computer in the queue or until the queue is empty. If the queue is empty then the target computer is not infected, else if we find the target computer in our queue at some point of time then that means there is a path from infected computer to this computer so it will get infected as well.

Algorithm:

Input : A queue and a visited array visited[ ]. (C_a), (t_a), (C_b), (t_b).
Solution : Create a graph G=(V,E) in which V are the computers and E denotes the communication time. Find the (C_a) by doing traversal of the graph. Push this node <(V_i),(t_a)> into queue. Let edge(u,v) denotes the communication time between (u,v). Node contains 2 values : vertex value and time of communication from its immediate parent.

```
while(queue is not empty)
    node u = q.pop()
    visited[u] = 1
    if(u == $C_b$)
        return "target computer will be infected".
    For all edges (u,v) such that visited[v] == 0 and edge(u,v) <= ($t_b$) and
    edge(u,v) >= (timestamp(u))
        q.push(node<v,timestamp(u)>);
if (queue == empty)
    return "target computer will not be infected".
```

Ans 4. ***Asymptotic Complexity***

$\sum_{i=1}^{n}(i^2+5i)/(6i^4+7) \quad \equiv \sum_{i=1}^{n} 1/i^2 \quad \prec lglgn \quad \prec \sqrt{lgn} \quad \prec \lg\sqrt{n} \quad \equiv \text{lnn} \quad \equiv \sum_{i=1}^{n} 1/i$

$\prec 2^{\sqrt{lgn}} \quad \prec lgn^{\sqrt{lgn}} \quad \prec min(n^2, 1045n) \quad \prec ln(n!) \quad \prec n^{ln4} \quad \prec \lfloor n^2/45 \rfloor \quad \equiv n^2/45$

$\equiv \lceil n^2/45 \rceil \quad \prec 5n^3 + logn \quad \prec \sum_{i=1}^{n} i^{77} \quad \prec 2^{n/3} \quad \prec 3^{n/2} \quad \prec 2^n$

Ans 5. *Euler Tour*

a) Euler Tour detection :

i) Mathematical proof : Euler tour is a tour which traverses every edge exactly once in a connected graph. So it is of the form like : (v_0, v_1, v_2, ..., v_n, v_0). So in this Euler tour, degree of a vertex (except v_0) is the number of times (c) it appears in this sequence multiplied by 2 since there will be an edge from where we will come to this vertex and an edge to go out from that vertex. Let's say that vertex to be u.

Degree of vertex u = 2*(c), where u can belong to v_1, v_2, ..., v_n.

Also, if u == v_0 then since v_0 is starting and ending vertex of Euler tour, so its degree should be 2. So this justifies that each vertex in Euler tour must have an even degree.

ii) By Contradiction : Suppose there is a vertex of odd degree, then while traversing from that node, one edge should be used to come to this vertex and one edge should be used to go out from that vertex. But as this vertex has odd degree, either one edge will be used twice or one edge will not be used at all in the Euler tour which violates the property of Euler tour. So as we can see every node should have even degree to satisfy the property of an Euler tour.

b) Euler Tour find:

First we will detect whether Euler tour is possible or not using the property that all vertices should have even degree and graph should be connected as well. If Euler tour is possible, we will select a random vertex(u) from a graph and do depth first traversal using stack. We will push any one neighboring vertice(v) of that vertex(u) to the stack and remove the edge from u to v as we have to traverse each edge exactly once. When popping the element from stack we will put that element into our Euler path. We will keep doing this until stack is empty.

Algorithm :

```
if graph is not connected
        print "Euler tour not possible".
if graph contains odd degree vertices
        print "Euler tour not possible".
path = empty
```

```
select a random node and push it into stack.
while(stack is not empty)
    u = stack.top()
    if (u has neighbors)
        for any one neighbor v of u
            remove edge(u,v)
            push(v) into stack
    else
        item = stack.pop()
        path.append(item)
return path.
```

Ans 6. ***Proof : Acyclic Connected Graph has atleast 2 vertices with degree 1***

Proof by Induction :

i) Base Case : Let's say we have only 2 vertices u and v. Since the graph is connected and acyclic, so there must be a single edge between them. So this means that both have degree 1. In this case, we have 2 vertices with degree 1 so our base case is valid.

ii) Induction Step : Let's say this is valid for k vertices which means there are atleast 2 vertices (u and v) in a graph of k vertices which have degree 1.
Proof for (k+1) vertices : We add one more vertex (say m) to the above graph of k vertices. There are 2 cases possible:
a) This vertex is in the middle of graph somewhere. So we still have 2 vertices, u and v which has degree 1.
b) This vertex is inserted at the end of the graph. So it will replace any of (u,v) with m and still have atleast 2 vertices (either u and m or v and m) with degree 1.

This proves that acyclic connected graph has atleast 2 vertices with degree 1.