Solution submission of Assignment 1

Divyansh Upreti (Student Id: 112026646)

Solutions

1. The jar stress testing problem:

   (a) **Budget of $k = 2$ jars:** Consider for n rungs and k jars : We will divide the n rungs for kth jar in such a way that kth jar will have same number of rungs to work with as the (k-1) jars. This can be possible only when we allot: $n^{1/k}$ rungs to each jar. So in the case of two jars, we will allot $\sqrt{n}$ rungs to each jar. So we keep on moving up in multiples of $\sqrt{n}$ up until jar breaks. Let's assume that it brekas at $i * \sqrt{n}$ rung, so the next safest rung will be between $(i-1) * \sqrt{n}$ and $i * \sqrt{n}$. So we drop the second jar from $1 + (i-1) * \sqrt{n}$ rung upward. Hence the upper bound of dropping first jar will be $\sqrt{n}$(when n is perfect square) and $2 * \sqrt{n}$(when n is not a perfect square - this can be proved mathematically). Same $\sqrt{n}$ number of operations will be true for second jar as well. Considering $f(n)$ to be $\sqrt{n}$ ignoring the constant terms, we can say that $\lim_{n\to\infty} \frac{f(n)}{n} = 0$.

   (b) **Budget of $k > 2$ jars:** We can apply the above principle here as well. For generalized k, to balance out the number of rungs for each glass jar, we should ideally allocate around $n^{1/k}$ rungs for each jar. However, to calculate the approximate number of trials required after dropping first jar, we can use this using expression: $n/x = n^{1/k}$ which gives $x = n^{k-1/k}$. Thus after first jar, we will repeat our process for $n^{k-1/k}$ rungs recursively for the value of k. Thus, in total we will get the upper bound as $\mathcal{O}(n^{1/k})$. To prove for functions $f_1, f_2, f_3, ...$, $\lim_{n\to\infty} \frac{f_k(n)}{f_{k-1}(n)} = 0$, we can substitute worse case time complexities function $(n^{1/k})$ in expression $\lim_{n\to\infty} \frac{f_k(n)}{f_{k-1}(n)}$ which evaluates to $\lim_{n\to\infty} \frac{n^{1/k}}{n^{1/k-1}}$ and this evaluates to 0 for each k. So our strategy of allocation of $n^{1/k}$ rungs for each glass jar is valid for all k.

2. This problem can be solved using properties of bipartite graph. Here the solution will be achieved by colouring the specimen with same colour (red) if they belong to same specimen otherwise coloring with different (blue) if they belong to different specimen. We will consider a graph $G(V, E)$ where V denotes specimen and E denotes the relationship between specimen. Algorithm to verify consistency of results of the problem's experiment is follows:

Algorithm 1 checkConsistency(G(V,E))

```
Start with any vertex s and colour it red.
Mark s as visited.
Put s in 0th layer.
for i in 0,1,2,... do
   Take each vertex u in ith layer and for each edge (u,v):
   if  v has not been visited  then
      Mark v as visited
   end if
   if  judgement of (u,v) is 'same'  then
      Colour v with same colour as u
   else
      Colour v with opposite colour of u
   end if
   end for
   for each edge(u,v) do
      if the judgement was 'same' then
         if u and v have different colours then
            Inconsistency in experiment
         end if
      else if the judgement was 'different' then
         if u and v have same colours then
            Inconsistency in experiment
         end if
      end if
   end for
   Experiment is consistent
```

The running time of this algorithm will be $\mathcal{O}(V+E)$ where V is the number of speciments and E is the number of edges between them.

3. The problem can be solved using BFS technique. We will make a graph $G(V, E)$ where V will have all nodes in the form (C_n, t_n). For each pair of nodes (form : (C_i, C_j, t_n)) in the input set, we comprise our set V by having an edge between (C_i, t_n) and (C_j, t_n) and vice versa to denote a connection between C_i and C_j at time t_n. We also will have a connection from each (C_i, t_a) to (C_i, t_b) in graph for each C_n where $t_a <= t_b$ to specify that t_a was the first time that C_i got infected. Now, given a time x where virus was introduced at a computer C_a, we will find a node (C_s, t_s) in our set V where t_s will be greater than or equal to x. We need to find if C_a infected C_b by time y. So, we start at (C_s, t_s) and using BFS, traverse the neighbours one by one until we find a node (C_b, t_b) where $t_b <= y$ (in this case our answer will be true) or the graph has been completely traversed (in this case our answer will be False). The running

Algorithm 2 checkInfection(G(V,E),C_a,C_b,x,y)

1: Traverse all nodes of graph and find out node (C_s, t_s) (say *node*) such that t_s is greater than or equal to x
2: Mark *node* as visited and put in 0th layer
3: **for** `i in 0,1,2,...` **do**
4: `Take each node u in` i^{th} `layer and for each edge (u,v):`
5: `if v has not been visited then`
6: `Mark v as visited`
7: `if v=`(C_b, t_b) `where` $t_b < y$ `then`
8: C_b `will be infected by` C_a `by time` y
9: `return`
10: `end if`
11: `end if`
12: **end for**
13: C_b won't be infected by C_a by time y

time of this algorithm will be $\mathcal{O}(E)$ where E is number of edges in $G(V, E)$.

4. The functions have been arranged in increasing asymptotic order below:
$\sum_{i=1}^{n}(i^2+5i)/(6i^4+7) < \sum_{i=1}^{n} 1/i^2 < \sqrt{\lg n} < \lg\lg n < \lg\sqrt{n} < \ln n \equiv \sum_{i=1}^{n} 1/i < (\lg n)^{\sqrt{\lg n}} < \min\{n^2, 1045n\} < \ln(n!) < n^{\ln 4} < \lfloor n^2/45 \rfloor \equiv \lceil n^2/45 \rceil \equiv n^2/45 < 5n^3 + \log n < \sum_{i=1}^{n} i^{77} < 2^{\sqrt{\lg n}} < 2^{n/3} < 3^{n/2} < 2^n$

5. (a) Assume for a connected graph G, all vertices have even degree. Let's start at any arbitrary node s and start going through edges of G such that no edge is traversed more than once. In this way, we will surely land on the starting vertex s at some point of time(as all vertices are of even degree). At this point of we may have traversed all edges of G and thus we can say that G has an Euler tour (a closed walk through G that traverses each edge exactly once). If there are some edges still remaining, then starting from s, we again start traversing the unexplored edges till we again land at s after some point of time(as all vertices are of even degree). Let the previous walk be W_1 and second walk be W_2 and thus we can say that graph G has an Euler tour comprised of walks W_1 and W_2 that starts and ends at the same vertex s. We can generalize this for walks $\{W_i, i = 1, 2, ...\}$ till the complete Euler graph has been explored and we can safely say that the start and end vertex s will be same after the complete traversal for graph G. For a graph G with any vertex having odd degree, it may be true that the terminating vertex of traversal be other than s as there is no requirement to enter and leave vertex equal number of times when degree is odd. We can say that the traversal will start and end at the same vertex if and only if each vertex has even degree. Thus, we can say that a connected graph G has an Euler tour if and only if every vertex has even degree.

   (b) As evident from (a), an Euler circuit in a conneted graph exists iff each vertex has even degree. So, to check if graph $G(V, E)$ contains an Euler tour or not, we will check:

      i. G is a connected graph
      ii. Each vertex of G has even degree

   If both conditions are true, then G will have Euler tour. Algorithm to check if G has Euler tour:

Algorithm 3 checkForEulerTour(G(V,E))

Check if the graph is connected or not by algorithm `isConnected(`$G(V, E)$`)` given after this algorithm.
if isConnected(G) **then**
    for `each vertex v in V:` **do**
        if $deg(v)$ is odd **then**
            G does not have Euler tour
            return
        end if
    end for
    G has Euler tour.
end if
G does not have Euler tour.

Algorithm 4 isConnected(G(V,E))

Take any arbitrary vertex s
Mark s as visited
Call $DFS(G, s)$ algorithm specified after this algorithm
for `each vertex v in` G **do**
    if v is not visited **then**
        return False
    end if
end for
return True

Algorithm 5 DFS(G(V,E),s)

Mark s as visited
for `each vertex v adjacent to s` **do**
    if v is not visited **then**
        Call $DFS(G, v)$
    end if
end for

Running time of $DFS()$ will be $\mathcal{O}(V + E)$; for $isConnected()$, the running time will be $\mathcal{O}(V)$; for $checkForEulerTour()$, the running time will be $\mathcal{O}(V)$ and $\mathcal{O}(V + E)$ for $isConnected()$ method in it, so we can say that the running time of the complete algorithm will be $\mathcal{O}(V + E)$

6. Let's denote a connected acyclic graph wih n>=2 vertices by $G(V, E)$. Let's start depth first traversal with any aribitrary vertex s and visit its adjacent vertex. if G has n=2 vertices, then this adjacent vertex will be the end point of traversal and since there are no cycles we can say that both s and current vertex have a degree 1. If G has n>=2 vertices, then the adjacent vertex must have degree atleast 2 to move forward in traversal and this holds for all other intermediate vertices in process. So, these vertices cannot have a degree of 1. Also, because there are no cycles in G, we will never get back to an already visited vertex, and assuming the vertices are finite, the traversal will defintely end up at some point. Hence, we will finally

reach a unvisited vertex from where any new forward traversal will be impossible because there won't be any visited vertex adjacent to it; G being acyclic. This will be the second vertex having degree of 1 in G. There can be many such terminating vertices however and we can include them in our results but we have proved that there has to be a second vertex where graph G ends if graph is acyclic and connected with n>=2 vertices. Thus, we have proved that a connected acyclic graph with n>=2 vertices will have atleast 2 vertices with degree 1.