Assignment 1: CSE548: Analysis of Algorithms

Shivasagar Boraiah — #112077826

September 16, 2018

Problem 1. Textbook [Kleinberg & Tardos] Chapter 2, page 69, problem #8.

Proof. (a)
Input: K=2 (jars), N floors.
Output: To describe a strategy to drop a jar, f(n) that grows slower than linearly.

- If there is only one jar, we go to each floor and verify for the stress by dropping the jar. Worst case, we try all N floors.
- Since we have 2 jars, if we plan to shift X intervals of floors every time, the first floor to experiment is X. And, if the jar breaks, then we need to linearly proceed from floor 1 to floor X-1 to determine the stress. The maximum trails will be 1+(X-1)
- If the jar handles stress at X^{th} floor, then our next trail should be at X+(X-1). If a jar breaks, then we have to linearly try handling the stress from X+1 to X-2. If we follow the same strategy, the floor number we cover with every X interval is, X, (X-1), (X-2),..., 3, 2, 1.
- The total numbers of floors is nothing but the sum of floors covered in each interval. i.e N = X + (X-1) + (X-2) + ... + 3 + 2 + 1 = X*(X-1)/2.
- The quadratic equation can now be solved to obtain $f(N) = (-1 + (1 + \sqrt{1 + 8N}))/2$
- For very large value of N, the function f(N) grows in the order of $O(\sqrt{N})$.

So, for large N, f(N) grows slowly compared to N itself. so, f(N) $\in$ o(N). Or, $\lim_{x\to\infty}(f(N)/N) = 0$
□

Proof. (b) I have discussed this problem with Abhishek Reddy Y N.
Input: K jars, N floors.
Output: To describe a strategy to drop a jar, such that fk(N) grows slower than fk-1(N).

- For a given K jars, if we drop a jar from floor X and jar breaks, then we should check for X-1 floors with K-1 jars.
- For a given of K jars, if we drop a jar from X and jar did not break, then then we need to check for next N-X floors with K jars.

- from (a), if we have one jar then we have to drop it a maximum of N times. If we have 2 jars, then at worst case we have to drop each jars maximum of $N^{1/2}$ times and the total drops are 2(2-1)$N^{1/2}$ times. Similarly, If we have 3 jars, then we can get to drop each jar $N^{1/3}$ times and total count comes to 2(3-1)$N^{1/3}$. So, if we are given with (k-1) jars then the generic number of times we can drop a each jar at intervals of $N^{1/k-1}$ and total drops can be minimized to 2(k-1)$N^{1/k-1}$. Clearly, given k jars and N floors, the function fk(N) = 2k$N^{1/k}$ grows slower than fk-1(N) =2(k-1)$N^{1/k-1}$

□

Problem 2. Textbook [Kleinberg & Tardos] Chapter 3, page 107, problem #4 .

Proof. Input: Set of Edges, E weighted with "same" or "different" for the set of Vertices's (specimens), V.
Output: After labeling the set V with labels 'A' or 'B'. Compare those Vertices's set (u,v) belongs to V with the Edge set, E and conclude the algorithm is consistent or not.
Algorithm brief: With the Edge set, E, design a Graph, G and run a BFS(G) and label the each vertex "A" or "B" depending on the edge weight. Once all the vertices's are marked, compare the Queue, A obtained from BFS with Egde set, E for all edges, (u, v). If the vertices's of A and their respective edge holds consistency then the Edge set, E is definitely consistent. The below algorithm (Algorithm 1) runs in a linear line of O(V+E).

Data: Edge set, E weighted with "same" or "different".
Result: To verify whether the Edge set, E is consistent with this vertices's or not.
put (NULL, Start) into the Q;
label(Start) := A;
consistent := 0;
while *Q != NULL* **do**
  take (P, V) from the Q;
  if *V is not marked* **then**
    Marked(V);
    if *edge(P, V) == "same":* **then**
      A[label(V)] := A;
    else
      A[label(V)] := B;
    end
  else
  end
  for *each edge (v, w)* **do**
    put vw into the Q
  end
end
for *edge(U, V) from edge-set E* **do**
  if *(edge(U, V) == "same": and A[label(U)] == A[label(V)]) or (edge(U, V) == "different" and A[label(U)] != A[label(V)]):* **then**
    consistent := 1;
  else
    consistent := 0;
  end
end
if *(consistent):* **then**
  Algorithm is consistent;
else
  Algorithm is Not consistent;
end

Algorithm 1: Algorithm to check the consistency of Vertices's, V (Specimens, N)

□

Problem 3. Textbook [Kleinberg & Tardos] Chapter 3, page 107, problem #11.

Proof. I have discussed this problem with Abhishek Reddy Y N.
With the Ordered tuple set we can generate a directed Graph G, with (Ci, Cj) as vertices and time, tx being the edge. With the Graph, G we can construct an Adjacency list, A associated with each computer. With the set of triplets, we want to decide whether a virus introduced at computer Ca at time x will effect the computer Cb at time y. To achieve this, Lets traverse the Ca list to find an edge which has greater time than x and lesser time than y. i.e, min {y, max{x, edge[Ca, Cb]}}. So, after this, lets traverse all

the nodes that are directly connected to Ca and print the info if they are infected. We should repeat this for all possible children of Ca. For set of nodes that are not in the infected array, they must have not effected by Ca because they have either communicated before x or disjoint with Ca list. This strategy can be achieved in O(V+E) time.

Data: Edge set, E with the time that two vertices's communicate
Result: To verify whether the node Cb at y connected to Ca will be infected if a node Ca at x is infected.

```
DFS(Ca, x, Cb, y);
for (edge (Ca, Ci) in Adj[Ca]) do
    marked(Ca);
    if ((x <= edge(Ca, Ci) <= y) && node(Ci) is !marked()) then
        Cb is infected by the time y;
    else
        DFS(Ci, x, Cb, y);
    end
end
```

Algorithm 2: Algorithm to check the infection of virus at Cb which is related to Ca

□

Problem 4. 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 $f(n) \equiv g(n)$) or if one is strictly less than the other ($f(n) \in o(g(n))$ and use the notation that $f(n) \prec g(n)$).

Proof. Function in the increasing asymptotic order

$\sum_{i=1}^{n}(i^2+5i)/(6i^4+7) \equiv \sum_{i=1}^{n} 1/i^2 \prec \lg\lg n \prec 2^{\sqrt{\lg n}} \prec (\lg n)^{\sqrt{\lg n}} \prec \sqrt{\lg n} \prec \lg\sqrt{n} \equiv \sum_{i=1}^{n} 1/i \equiv \ln n \prec \ln(n!) \prec \min\{n^2, 1045n\} \prec n^{\ln 4} \prec \lfloor n^2/45 \rfloor \equiv \lceil n^2/45 \rceil \equiv n^2/45 \prec 5n^3 + \log n \prec \sum_{i=1}^{n} i^{77} \prec 2^{n/3} \prec 3^{n/2} \prec 2^n$ □

Problem 5. An Euler tour of a graph G is a closed walk through G that traverses every edge of G exactly once.

Problem 5(a). Prove that a connected graph G has an Euler tour if and only if every vertex has even degree.

Proof. Suppose graph G has Euler tour.

- Let us inspect all the edges that incident on vertex, V of G.
- Every vertex, U of G when arrives at V, it has to enter from one edge and leave the vertex from another edge to satisfy the Euler tour.
- So, if V has K in-degree, the it must have K out-degree edges which totals to 2K edges of V, which is even.

This condition holds true for all the vertices's of G as all of them in the Euler tour.

I have refereed few on-line materials such as wikipedia, stack overflow to understand the theorem.
Lets prove by induction that, if every vertex has even degree then if can have Euler path.

- Base Case: S(2): For two vertices U, V in S, since the degrees are even there as to be a even number of edges between them. With 2K edges between U and V, Traversing U to V via K edges and going back with another set of K edges make sure that there could be a Euler path. So, S(2) is true.
- Induction Hypothesis: Lets assume that S(K) with even degree has Euler tour.
- Induction Step: $S(K)$ ==>S(K+1)
- Lets add Vertex W to S(K). Since it should have even degree, for every vertex connecting with W enter via one edge and exit via another which will only extend the Euler path. So, S(K+1) is true.

By Induction, S(N) is true. □

Problem 5(b). Describe and analyze an algorithm to compute an Euler tour in a given graph, or correctly report that no such tour exists.

Proof. To compute Euler tour, let us traverse the graph from start node to all connected nodes. For a graph to satisfy Euler tour, every vertex should have even degree and start and end of the traverse should be same. The algorithm can be computed in O(E) time, for a graph G with edge set, E.

```
Data: Graph G
Result: To verify that Graph G has Euler tour
put (NULL, Start) into the Q;
start = Start;
while Q != NULL do
    take (P, V) from the Q;
    end = V;
    if degree(V) % 2 == 0: then
        Marked(V);
    else
    end
    for each edge (V, W) do
        put VW into the Q
    end
end
if start == end then
    Euler tour found;
else
    Euler tour not found
end
```

Algorithm 3: Algorithm to check for Euler tour

□

Problem 6. Prove that any connected acyclic graph with $n \geq 2$ 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.

Proof. Proof by Induction:

- Base case: S(2): for first and last or the two vertices of S, U and V to be connected and acyclic, there has to be one edge exists between the two vertices and degree(U)=degree(V)=1. So, the the proof is true for two vertices.
- Induction Hypothesis: Let us assume that S(K) is true for K connected, acyclic vertices where the first and last vertices U and V has degree 1.
- Induction step: $S(K)$ ==>S(K+1)
- New vertex, W to be added to the graph of K vertices. Since the graph should be connected, we should introduce vertex(W) to one of the already existing vertices in the graph. For the graph to remain acyclic, we have to add a connecting edge to W from S that does not form a cycle. If you add an edge from Vertex(U) or Vertex(V) to Vertex(W), then atleast two vertices will have odd degree. If you add Vertex(W) to any other vertices of S with even degree, still the hypothesis holds true.

So, S(K+1) is true. This implies, Using Induction, S(N) is true. □