CSC263 Assignment 4

Akhil Gupta, 1000357071

November 2015

1. Now, suppose you are given a group of N statements, numbered from 1 to N. Each statement has the form: "Statement X is TRUE/FALSE, where X is a number between 1 and N. Your task is to figure out whether this group of statements forms a paradox. In particular, answer the following questions.

   (a) Describe how to construct a graph to solve this problem. Be precise: state clearly what vertices your graph contains (and what each vertex represents) and what edges your graph contains (and what each edge represents).

   Answer: We use a directed weighted graph to solve this problem. Our graph contains N vertices and each vertex represents a statement n of the form "Statement X is TRUE/FALSE" where X is a number between 1 and N. There will be one directed out-edge from every vertex and that edge represents the claim made by that statement. The graph will be stored as an adjacency list where the linked list stored as each node will be descendants(n_i). We also have 2 properties:
   1) For every statement n_i let descendants(n_i) be the statements that comment about n_i, and by fixing the state (true or false) of n_i fixes the state of descendants(n_i) and n_i.
   2) If there is a valid assignment of states to statements, then the n statements do not form a paradox, and if there exists no such assignment of states, then the n statements form a paradox.

   (b) Give a necessary and sufficient condition for the N statements to form a paradox. Justify your answer.

   Answer: n statements will form a paradox if and only if there is a cycle from N_1 to N_1 where the path length from N_1 to N_1 has an odd number of false implications.

   (c) How do you efficiently detect whether your graph from part (a) satisfies your condition described in part (b)? Describe your algorithm in concise (but precise) English.

   Answer: Our algorithm starts by assigning/storing a state (truth value: true or false) to a random vertex N_i and assigning a weight of 0 to an edge from n_i to n_j if statement n_i says that statement n_j is true, and a weight of 1 to an edge if statement n_i says that statement n_j is false. Then we check whether the path length from vertex N_i to vertex N_i on the graph formed by descendants(n_i) using Depth-First Search. If the path length is odd, then the n statements for a paradox.

   (d) Analyse the worst-case runtime of your algorithm.

   Answer: According to lecture, DFS runs in $O(|V| + |E|)$ time. In our case, our algorithm starts by assigning a truth value to a random vertex and assigning weights to the edges which takes $O(1)$ time, then it performs DFS twice (once for true and once for false) on descendants(n_i) which takes $2O(|N| + |n|)$ time. Our algorithm also has a counter which keeps track of the visited nodes and the corresponding edge-weights associated with that node. Then it checks whether it is odd or even. All this takes $O(1)$ time. Therefore, in the worst-case, our algorithm runs in $O(1) + O2(|N| + |n|) + O(1) = O(|N| + |n|)$ time.

2. Now, you must devise an algorithm BucketMeasure(m, n, k), which takes m, n and k as inputs and outputs a sequence of moves that results in one bucket (it does not matter which one) having exactly k litres of water. The number of moves in the returned sequence must be the smallest possible number of moves that are needed to achieve the goal. If it is impossible to measure k litres of water using the two buckets, the algorithm returns .

Answer the following questions.

(a) How do you construct a graph for solving this problem? Describe the vertices and edges in your graph clearly and precisely.

Answer: We use a directed graph to solve this problem. Our graph is in the form of a grid, where the x-axis is 1 litre increments of bucket A with m litres and y-axis is 1 litre increments of bucket B with n litres. Each point in the grid can be represented by a node in the graph. Each node contains the ordered pair representing the amount of water one can hold in each bucket (m, n). The directed edges of the graph represent transitions from one state to another and which states are possible in one move from the current state to the new state. The graph will be stored as an adjacency list where the linked list at each node stores the list of possible moves from that node. Each move can only perform the following 3 operations:
1) Fill a bucket until it's full,
2) Empty a bucket,
3) Use the water in one bucket to fill the other until one of the buckets is full or empty.
The initial state of both buckets is $(0, 0)$.

(b) How does your algorithm work? Give a detailed description and justify its correctness.

Answer: We need to devise an algorithm BucketMeasure(m, n, k) which takes m, n and k as inputs and outputs a sequence of moves that results in one bucket having exactly k litres of water. Our algorithm starts by creating the above-described graph and the initial states of both buckets as $(0, 0)$. In order to get the smallest possible number of moves, we simply perform a Breadth-First Search (BFS) on the graph. We search from the original state to a state that has k as one of the members of the ordered pair (m, n) i.e. either $m = k$ or $n = k$. BFS computes d[v] which stores the distance value between 2 nodes, so in our case, the smallest possible numbers of moves is the distance value from $(0, 0)$ to a state that has k in the ordered pair.The algorithm:
1) Our algorithm builds a graph of all possible edges starting at $(0, 0)$.
2) Using Breadth-First Search, it goes through all the edges once and stores d[v] at that node.
3) The algorithm then outputs a series of nodes/moves from the original state $(0, 0)$ to a state where k is one of the members of the ordered pair (m, n) by checking the smallest d[v] value of a node where k is a member of that node.
4) If none of the nodes have k as a member of the ordered pair, it is impossible to measure k litres of water using two buckets, so the algorithm returns NIL.

(c) Analyse the worst-case runtime of your algorithm.

Answer: According to lecture, BFS runs in $O(|V| + |E|)$ time. In our case, the number of vertices are $m+1$ * $n+1$ since $(0, 0)$ is also a vertex shared by both m and n. The number of edges is less than $|V|^2$ i.e. $\leq ((m + 1) * (n + 1))^2$. In the worst-case, our algorithm goes through all possible edges/moves, therefore, worst-case runtime is $O(((m + 1) * (n + 1)) + ((m + 1) * (n + 1))^2)$ which is atleast $O((m + 1) * (n + 1))^2$.