CS 3510 Honors Algorithms
Solutions : Homework 2

Problem 1. [Running SCC]

(a) We can try out by constructing G^R, which has the edges in reverse direction as compared to G. When we do DFS in G^R, starting from the vertex C, we will see that we reach vertices C, D, A, B, H, G, in that order. Then we see that we have to start exploring from an unexplored vertex. Picking K and exploring, we reach I, E, J, F, L in that order. The pre/post numbers are as in the below table.

		A	3/6	B	4/5		
C	1/12	D	2/7	E	15/20	F	17/18
G	9/10	H	8/11	I	14/23	J	16/19
		K	13/24	L	21/22		

(b) In the first iteration of the loop, the algorithm picks K. The connected component K, L, I is removed from G'. Then vertex E is picked. The vertices E, F, J form an SCC and are removed. Then vertex C is picked and the SCC C, G, H are removed. Then D is picked, and all of the remaining vertices D, B, A form another Strongly Connected Component. Then no vertex remains in G' and the algorithm exits from the *while* loop. (**Verify!**)

(c) The SCC's are output as mentioned in the last part. The resulting DAG is as follows. From the SCC (A, B, D) there are two directed edges going out to the SCC's (C, G, H) and (E, F, J). From each of the SCC's (C, G, H) and (E, F, J), there is a directed edge to the SCC (K, L, I). The SCC (A, B, D) is a source and SCC (K, L, I) is a sink.

Problem 2. [Timing with Hourglasses]

(a) First of all, we make an interesting observation. The claim is that we can measure a time interval of t minutes **if and only if** it can be represented in the form $t = 11a+7b+5c$, where $a, b, c \in \mathbb{Z}$ (ie., a, b, c are integers, positive or negative). [**Exercise :**Try and prove this claim, note that you have to prove both directions.]

So note that any possible measurement of 13 minutes could be represented in terms of a, b, c where $13 = 11a + 7b + 5c$. If a, b, c are all positive, it is very easy to see how we can measure the required time. If we have to measure time of the form $f - g$, where f and g are both positive, we can measure g and start the measurement from the time when g gets over till the time when f is over. Observe that $|a| + |b| + |c|$ corresponds to the total number of turns of the hourglasses. So the problem is to find (a, b, c) such that $11a + 7b + 5c = 13$ and $|a| + |b| + |c|$ is minimized.

We could have a graph with nodes corresponding to triplets (a, b, c). From each node in the graph, we could move to the node where (atmost) one of a, b, c differs from its previous value by 1. For example, $(5, 4, -3)$ has edges to the nodes $(4, 4, -3)$, $(6, 4, -3)$, $(5, 3, -3)$, $(5, 5, -3)$, $(5, 4, -2)$, $(5, 4, -4)$. (This construction is to ensure that every edge corresponds to one turn of an hour glass.) The time measured by a triplet (a, b, c) is given by $11a + 7b + 5c$. The problem can be restated in terms of this graph as follows.

> Given graph G of triplets (a, b, c), where there exists an edge from (a, b, c) to (a', b', c') if and only if $|a - a'| + |b - b'| + |c - c'| = 1$, find the node closest to the point $o = (0, 0, 0)$, such that $11a + 7b + 5c = 13$.

(b) Since the problem is to find the closest node which has a certain property, we could use BFS starting from $(0, 0, 0)$ to compute the answer. Note that as we traverse each node, we check to see if the sum $11a + 7b + 5c$ is 13. We will terminate at the first node which gives us the required sum. Since we are traversing the nodes in the order of proximity to $(0, 0, 0)$, BFS assures us that $|a| + |b| + |c|$ is minimized.

(c) Constructing the graph as we proceed, starting from $(0, 0, 0$, we can see that none of the nodes which are at a distance 1 or 2 measure 13 minutes. But the triplet $(1, 1, -1)$ is at a distance 3 from $(0, 0, 0)$ and measure 13 minutes. This corresponds to a measurement as follows.

> Start the 7 minute and 5 minute clocks (2 turns). After the 5 minute clock is done, start measuring time. Start the 11

minute clock immediately after the 7 minute clock is over(1 turn). The measurement stops when the 11 minute clock stops. Total time measured is $(7-5)+11=13$ minutes.

Problem 3. [**Problem Set Multitasking**]
First note that we can hire as many friends as we want. First we model this in a graph, where there is a directed edge from problem i to problem j if and only if the problem j depends on problem i. Assuming we have enough people to go through all the nodes, we have to find out when is the earliest time in which we can complete each problem. Note that we cannot have a cycle in this graph, else we will have a cycle of dependencies which will render the problem set insolvable. (**Think!**)

To find out a schedule that completes all the problems in the shortest time, we need to first sort the problems in a topological order. This will make sure that we get the nodes in the order of dependencies. ie., If problem i depends on problem j, then the sorting will ensure $V[j] < V[i]$, where $V[j], V[i]$ are the nodes corresponding to the problems i, j. Then we will determine when the problems can be done. We are representing the graph G as an array of vertices V, where each $V[i]$ has an associated field $V[i].time$, which denotes the earliest time in which we can do the problem.

```
Schedule(Array structure V representing G)
      Array V' = TopologicalSort(V)
      For i=1 to n {
          earliesttime = 1;
          For all nodes V'[j] with an edge pointing to V'[i] {
              if (earliesttime < V'[j].time + 1)
                  earliesttime = V'[j].time + 1;
          }
          V'[i].time = earliesttime;
      }
      Return V'[i].time for all i
      Return Maximum (over all i) V'[i].time
```

Now we are left to show that the $V'[i].time$ is computed correctly. We will use induction to show the same. When there is only problem, the algorithm returns $V'[i].time = 1$, so it is correct. For inductive step, assume for all $j < i$, $V'[j].time$ is computed correctly. The earliest time that we can do problem $V'[i]$ is 1 plus the time needed to complete all the problems $V'[j]$ on which problem $V'[i]$ depends. Since we have ordered the graph topologically, we would have computed all the $V'[j].time$ values correctly before we come to $V'[i]$. Thus $V'[i].time$ is calculated correctly.

Problem 4. [**U.S.Geography**]

(a) The articulation points are the ones, on removal of which the number of connected components of the graphs increase. The articulation points in this graph are NY and NH.

(b) The only valid 4-tuples are (AZ,UT,CO,NM) and (WI,MI,IN,IL).

(c) A way of going about it is to search from all the vertices w for a 4-cycle (w, x, y, z), and then check for the no-edge conditions.

```
4tuplefind(G)
For all vertices w in G {
    For all neighbors x of w {
        For all neighbors y of x {
            For all neighbors z of y {
                if ((w,z) is an edge AND
                    (w,y) is not an edge AND
                    (x,z) is not an edge)
                    Output (w,x,y,z);
}  }  }  }
```

This is a brute force algorithm which checks all possible 4-tuples. Assuming n vertices, the first *for* loop takes n iterations. Each of the three inner *for* loops run for atmost d iterations, where d is the maximum degree. So the total running time is $O(nd^3)$.