CS180 Homework 3

Due: 4:00pm, 2/1/2017

1. Given a sequence $x_1, x_2, \ldots x_n$ of real numbers (not necessary positive), design an algorithm that find the consecutive subsequence $x_i, x_{i+1}, \ldots x_j$ such that the sum of numbers in it is maximum over all consecutive subsequences. [25pts]

   The idea of the algorithm is to store the current maximum sum and global max sum. We keep adding the numbers from left to right. If the sum is larger than the maximum sum we have seen so far, we save the beginning an ending index for current summation. If the current sum is negative, we just discard the sum and restart the summation from the next number. Note that it is possible we get a output $(n+1, n+1, 0)$ if all numbers are negative. If so, it means the subsequence we want is just empty. The time complexity of the algorithm is $O(n)$ since it use a linear scan of all numbers.

```
MaxSeq({x_1, x_2, ..., x_n})
  (start, end, sum)← (0, 0, 0)
  (left, right, max)← (0, 0, 0)
  for i ← i to n
      (start, end, sum)← (start, i, sum+x_i)
      if sum < 0
          (start, left, sum)← (i + 1, i + 1, 0)
      if sum > global max
          (left, right, max)← (start, end, sum)
  return (left, right, max)
```

2. Consider a scheduling problem for a project which consists of n jobs. Each job i has an execution time t_i, and a precondition that specifies job i can not be started before some jobs are finished. The precondition relations between jobs are acyclic. Multiple jobs can be executed at the same time if the preconditions for those jobs are satisfied. Give an algorithm to find a schedule that minimized the total time to finish all jobs. (Hints: we can think each job as a node in a DAG. An edge $i \to j$ represents job j can not be started before finishing job i.) [25pts]

   We reduce the problem to find the longest path in the DAG. For each job i we create a node i and if job i is the precondition of job j we create an edge $i \to j$. Let $G = (V, E)$ denote the graph.

```
Scheduling(G)
  Use DFS to get the topological sorting of graph G
  while there exists a node in the topological sorting
      choose nodes i_1, i_2, ... i_k with incoming degree in the topological sorting
      schedule them
      remove i_1, i_2, ... i_k their outgoing edges after they are finished
```

   To minimize the scheduling time, we just need to schedule as many jobs that can be scheduled at the moment. Hence, we use topological sorting to schedule the source jobs, then remove them and repeat this procedure. The time complexity of the algorithm is $O(|V| + |E|)$ because of doing topological sorting and removing all nodes.

3. Given a DAG $G = (V, E)$ and two nodes s and t in G, a $s - t$ path is a path that starts from s and ends at t. [50pts]

   (a) Find the subgraph of G induced by all the nodes on the $s - t$ paths (including s and t).
   The idea is run BFS on nodes s and node t and record the nodes marked in both BFS. The subgraph is induced by nodes marked in both DFS.

```
SUBGRAPH(G)
  Do to
  run DFS_1 on s and mark nodes by blue
  run DFS_2 on t against the direction of edges and mark nodes by red
  for each node v in G
      if v is marked by blue and red
      S ← S ∪ {v}
  return G_{s-t} as the subgraph of G induced by S
```

   The the first DFS marked the nodes that can be reached from s and the second DFS marked the nodes that can reach to t. Hence any nodes marked in both DFS are on some path from s to t. On the hand, any node on the path from s to t will be marked in both DFS. The time complexity is $O(|V| + |E|)$.

   (b) Two paths are different if they do not consist of the same sequence of edges. Give an algorithm to calculate the number of different $s - t$ paths in the subgraph.
   The subgraph G_{s-t} we find in the previous problem is also a DAG since a subgraph of DAG is a DAG. Moreover, if we do topological sorting on G_{s-t}, the source node will be s and the last node will be t. Hence, we can update the number of paths from the source $s$to t. The number of paths from node s to node v will be the summation of the number of paths from node s to node v's incoming neighbour plus the number of incoming neighbours.

```
NUMPATH(G, s, t)
  Use DFS to get the topological sorting of graph G_{s-t} = (V, E)
  Let v_1, v_2, ... v_n be the sorted nodes where n = |V| and v_n is node t
  for i ← 1 to n
      N(v_i) ← 0
  for i ← 1 to n
      N(v_i) ← Σ_k(N(v_k) + 1) where v_k is v_i's incoming neighbour
  return N(v_n)
```

   The DFS takes linear times. In the for loop, we do addition at most the number of total edges times. Hence, the time complexity of the algoirthm is $O(|V| + |E|)$.

   (c) In the subgraph, a node v is called separator if deleting v from the graph will disconnect all $s - t$ paths. Design an algorithm to decide whether there is a separator in the subgraph and find a separator if it there exists one. (Hint: Let v be a node on some $s - t$ path. Consider the number of $s - v$ paths, $N(s - v)$, and the number of $v - t$ paths, $N(v - t)$. What is the relation between $N(s - v) \cdot N(v - t)$ and $N(s - t)$?)
   In the previous algorithm, $N(v)$ saves the number of paths from s to v and so $N(v) = N(s-v)$. We can apply the previous algorithm with the reversed graph G and calculate the the paths from t to v, which is equivalent to $N(v - t)$. Then, for each node v we check whether $N(s - v) \cdot N(v - t) = N(s - t)$, if so node v must be a separator.

```
SEPERATOR(G, s, t)
    NUMPATH(G, s, t)
    for each node v in G
        N(s − v) ← N(v)
    G′ ←reverse G
    NUMPATH(G′, t, s)
    for each node v in G
        N(v − t) ← N′(v)
        if N(s − v) · N(v − t) = N(s − t)
            return v
    return none
```

The complexity of the algorithm is $O(|V| + |E|)$ since calculating $N(s - v)$ and $N(v - t)$ takes linear time and checking the condition for every node takes liner time. We prove the correctness of our algorithm by showing that the condition of $N(s-v) \cdot N(v-t) = N(s-t)$ is sufficient and necessary to check whether a node is a separator.

Lemma 1. *A node in the graph G is a separator if and only if* $N(s-v) \cdot N(v-t) = N(s-t)$

Proof. If a node is a separator, then every path from s to t has to pass through node v and a $s-t$ path can be decomposed into a $s-v$ path and $v-t$ path. Moreover, any combination of a $s-v$ path and a $v-t$ path will be a $s-t$ path. So we have $N(s-v) \cdot N(v-t) = N(s-t)$ If a node v satisfies the property $N(s-v) \cdot N(v-t) = N(s-t)$, then every path from s to v has to pass through v. Otherwise, there is a path that is not any combination of $s-v$ path and $v-t$ path and so, $N(s-t) > N(s-v) \cdot N(v-t)$. □