CS 260 - Fundamental Algorithm **Fall 2016**

HomeWork 1

October 4, 2016 *Liangjian Chen*

1. (Problem 2)

Algorithm 1 Find a circle in a graph

function FIND_CIRCLE(u, e')
  push u into stack
  for $e(u, v) \in E$ except e' **do**
    if v is in the stack **then**
      report v is in the circle
      while the top of stack(x) is not v **do**
        report x is in the circle
        pop x
      end while
      return find a circle
    else if v haven't been visited **then**
      if FIND_CIRCLE($v, e(u, v)$) finds a circle **then**
        return find a circle
      end if
    end if
  end for
  return do not find a circle
end function
Initialize: make a stack empty
FIND_CIRCLE(1,None)

2. (Problem 3)

   First, apply topological ordering algorithm into the graph. If algorithm delete all the node in the graph, it is a DAG. Otherwise, apply BFS into the rest of graph to find connected components. Every connected component is a circle. Both topological ordering and BFS is $O(m + n)$.

3. (Problem 4)

   It is a typical problem about disjoint set with distance. Let's define two butterflies is in same specie, if the distance between these two are even. Otherwise, the distance is odd. Like typical disjoint set, every butterfly is a set only consist itself at begining. The algorithm going through all m judgments and merge two set together. However instead of merging them directly, here we merge two sets by carefully choosing the distance(1 or 0) between two representatives to meet our definition(Odd or Even). When two butterflies are in the same set, we are able to check its consistency by checking their distance.

initialize disjoint set is $O(n)$. There are $O(m)$ disjoint set operations. Each of them costs $O(1)$. Total time is $O(n+m)$.
There is a same question in `http://poj.org/problem?id=2492`

4. (Problem 6)

   Assume that T's edges set is E_T, then for some extra set E_e, $E = E_T \cup E_e, E_T \cap E_e = \varnothing$. Now I want to argue that $E_e = \varnothing$. Assume that there is one edge $e(a,b)$ in set E_e, and define $d(x,y)$ as the distance between two any point x, y, which is the number of edges that the path between a,b contains in the tree T. Because T is BFS tree, so $|d(u,a) - d(u,b)| \leq 1$. Also, T is a DFS tree, so, one of a,b must be a ancestor of another. However, if both $|d(u,a) - d(u,b)| \leq 1$ and condition of ancestor hold, $e(a,b)$ must in the set E_T, it contradicts with assumption. So $E_e = \varnothing$.

5. (Problem 9)

   First, we can use BFS starting from u to find a path between u and v. Then, for every node x, record $d(x)$ which is the distance between x and u in BFS. We group the node by their $d(x)$, then among all groups, at least one group only has one node (Otherwise the total number of node would be larger than n). Then this node is the key node.(delete it there is not path between u and v). There is just a BFS in algorithm which cost $O(n+m)$.

6. (Problem 10)

   This problem could be solved by dynamic programming. Let $dp[x]$ represent the number of shortest path from u to x. Trans-equation is

   $$dp[x] = \sum_{\text{exist shortest paths } u \to .. \to y \to x} dp[y]$$

   Luckily, in this graph, every weight of edge is 1. So a BFS search framework can support this dynamic programming procedure.

   My algorithm goes through every node and every edges $O(1)$ time, so time complexity is $O(n+m)$.

7. (Problem 12)

   It is a problem of difference constraints. Let's assign two number to every single people(P_i), B_i and D_i which represent his/her time of birth and time of dead. Then we have three type of different difference constrain.

   (1) The dead time should be later than birth time for everyone. So first set of difference constraints, $D_i - B_i > 0$, which means $B_i - D_i \leq -1$.

   (2) For every constrain of kind of P_i died before P_j. The constrain is $B_i > D_j$ which is $D_j - B_i \leq -1$.

   (3) For every pair of P_i, P_j overlapping, the constrain is $B_i < D_j$ and $B_j < D_i$. which is $B_i - D_j \leq -1$ and $B_j - D_i \leq -1$.

Algorithm 2 Find the number of different shortest path

Require: a Graph $G(V, E)$two nodes v, w .

make queue Q empty
initialize all element in array dp zero.
put node v into Q
while Q is not empty **do**
  $x \leftarrow$ the front of queue
  pop the front of queue
  for $\forall e(x, y) \in E$ **do**
    if y has not visited **then**
      $dp[y] \leftarrow dp[y] + dp[x]$
      push y into Q
    end if
  end for
end while
return $dp[w]$

Recall the triangle inequality in shortest path problem, $d[v] - d[u] \leq c$ holds, if a edge from u, to v, cost c exist. Base on triangle inequality, we could build a graph by add a c-cost edge between u,v if we have a inequality $u - v \leq c$. Then use SPFA algorithm or Bellman-Ford algorithm to check whether there is a negative loop in this graph (In this problem, since all weights are negative, we could simply check whether there is a cycle instead of a negative circle). If there is, this system is not consist. Otherwise, we could assign arbitrary number to starting node, then inference all other number by the shortest distance between that node and starting node.