CSCI-UA.0310-004/005 Basic Algorithms May 11, 2015

Solutions to Problem 1 of Homework 11 (8 points)

Name: Jason Yao *Due: Wednesday, April 29*

(a) (5 points) Design $O(n)$ algorithm to test if a given *undirected* graph G is acyclic. Notice, the running time of your algorithm should not depend on the number of edges m!
(**Hint**: Could you argue fater termination of a regular DFS tester on undirected graph?)

Solution:

```
boolean testGraphForCycle (Graph G, Vertex v)
    S = stack.initialize()
    S.push(v)
    while (!s.isEmpty())
        v = S.pop()
        if (v.label == 'white')
            v.label = 'gray'
            for all edges from v to w in G.adjacentEdges(v)
                if edge is not visited && vertex is gray
                    // Cycle is detected, so exits
                    return true
                else
                    S.push(w)
            endFor
        endIf
    endWhile
    // No cycle is detected
    return false
END testGraph
```

□

(b) (3 points) Extend the above algorithm to actually print the cycle, in case G is cyclic.

Solution:

```
boolean testGraphForCycle (Graph G, Vertex v)
    S = stack.initialize()
    printQueue = queue.initialize()

    S.push(v)
    printQueue.enqueue(v)
    while (!s.isEmpty())
        v = S.pop()
        if (v.label == 'white')
            v.label = 'gray'
            for all edges from v to w in G.adjacentEdges(v)
                if edge is not visited && vertex is gray
                    // Cycle is detected, so exits
                    return true
                else
                    S.push(w)
                    printQueue.enqueue(w)
            endFor
        endIf
    endWhile
    // No cycle is detected
    while (!printQueue.isEmpty())
    print(printQueue.pop())
    return false
END testGraph
```

□

CSCI-UA.0310-004/005 Basic Algorithms	May 11, 2015

Solutions to Problem 2 of Homework 11 (10 points)

Name: Jason Yao — *Due: Wednesday, April 29*

Your job is to arrange n rambunctious children in a straight line, facing front, i.e., in the direction of the line. You are given a list of m statements of the form i hates j. If i hates j, then you do not want put i somewhere behind j, because then i is capable of throwing a pebble at j.

(a) (4 points) Give an algorithm that orders the line, (or says that it is not possible) in $O(m+n)$. time.

Solution:

We say that there are n vertex (children), with a list of m edges (statements) with direction. Direction is from some i to some j such that i "hates" j. This forms a directed graph G, and we can order this line via BFS, assuming no cycles inside of G.

```
void orderChildren(Graph G, vertex v)
    Q = queue.initialize()
    Q.enqueue(v)

    v.label = 'gray'
    while (!Q.isEmpty())
        v = Q.dequeue()
        for all edges from v to w in G.adjacentEdges(v)
            if (w.label == 'white')
                Q.enqueue(w)
                w.label = 'gray'
            endIf
            if (w.label == 'gray')
                // Cycle was found, breaks out
                return
        endFor
    endWhile
END orderChildren
```

□

(b) (6 points) Suppose instead you want to arrange the children in rows, such that if i hates j then i must be in a (strictly) lower numbered row than j. Give an efficient algorithm to find the minimum number of rows needed, if it is possible.

Solution:

```
int orderChildrenRows(Graph G, vertex v)
    Q = queue.initialize()
    Q.enqueue(v)
    rowArray [][] = ArrayList.initialize()
    maxSize = 0;

    v.label = 'gray'
    while (!Q.isEmpty())
        v = Q.dequeue()
        for all edges from v to w in G.adjacentEdges(v)
            if (w.label == 'white')
                Q.enqueue(w)
                w.label = 'gray'
                rowArray[i][0] = w
                if (maxSize == 0)
                    ++maxSize
            if (w.label == 'gray')
                rowArray[i][maxSize] = w
                ++maxSize
        endFor
    endWhile
    return maxSize
END orderChildrenRows
```

□

CSCI-UA.0310-004/005 Basic Algorithms	May 11, 2015

Solutions to Problem 3 of Homework 11 (12 (+6) points)

Name: Jason Yao — *Due: Wednesday, April 29*

Assume G is an undirected graph with weight function w, and $e_1 \dots e_m$ are the m edges of G sorted according to their weight: $w(e_1) \leq w(e_2) \leq \dots \leq w(e_m)$. Imagine you just ran the Kruskall's algorithm of G and it output an MST T of G. Now assume that somebody changes the weight of a single edge e_i from $w(e_i)$ to some other value w'. For each of the following 4 scenarios, describe the fastest algorithm you can think of to transform the original MST T of G to a new (and correct) MST T' of G after the edge weight change. Make sure you justify your answer, and express your running time as a function of m and n.

(a) (4 points) Assume $e_i \in T$ and $w' < w(e_i)$ (so we decreased an MST edge).

Solution:

A decrease an MST edge w_2 results in:

$w_1 \leq w_2$

MST stays the same

□

(b) (4 points) Assume $e_i \notin T$ and $w' < w(e_i)$ (so we decreased a non-MST edge).
(**Hint**: Compute the unique shortest path in T between the two end-points of e_i.)

Solution:

decrease a non-MST edge, w_i

old MST + edge w

=¿ creates cycle in MSY

=¿ find edge with MAX weight on this cycle =¿ remove it □

(c) (4 points) Assume $e_i \notin T$ and $w' > w(e_i)$ (so we increased a non-MST edge).

Solution:

increase a non MST edge

MST stays the same

□

(d) **Extra Credit:** (6 points) Assume $e_i \in T$ and $w' > w(e_i)$ (so we increased an MST edge).
(**Hint**: Try to find the smallest weight edge e_j which should replace e_i under the new weight.)

Solution: ***************** INSERT YOUR SOLUTION HERE *************** □

CSCI-UA.0310-004/005 Basic Algorithms	May 11, 2015

Solutions to Problem 4 of Homework 11 (11 points)

Name: Jason Yao *Due: Wednesday, April 29*

(a) (5 points) Let e be the maximum weight edge on some cycle of a connected graph $G = (V, E)$. Prove that there exists an MST T of $G' = (V, E\backslash\{e\})$ which is also an MST of G. Namely, some MST of G does not include e.

Solution:

if G has a cycle and w_i is the max-weight edge on this cycle =¿ there exists an MST of G such that w_i does not exist in the set of MST

case 1 MST(G) does not have w_i =¿ proved

case 2

w_i = fat edge on cycling

w_i exists in the set of MST(G) remove w_i =¿ 2 disjoint subtrees

look at cycle from u $\rightarrow$ t find edge (u_{new}, t_{new}), $u_{new} \exists V_1$, $t_{new} \in V_2$

$\rightarrow$ new MST has weight $\leq$ old MST

$(w_{new} \leq w_1)$

□

(b) (2 points) Consider the following idea for a greedy algorithm for finding some minimal spanning tree in an undirected weighted graph G:

(I) find a cycle in G.
(II) if there is no cycle, output G and terminate;
(II) else, if there is a cycle,
- find an edge e with maximum weight on this cycle;
- remove e from the graph G;
- return to step (I).

Prove correctness of this algorithm using induction and part (a).

Solution:

□

(c) (4 points) Describe details of the fastest implementation you can find for the algorithm in part (b) (or write a pseudocode), and analyze its complexity as a function of n (number of vertices) and m (number of edges).

Solution: ***************** INSERT YOUR SOLUTION HERE *************** □