CS180 HW2

Xilai Zhang

TOTAL POINTS

100 / 100

QUESTION 1

Problem 1 25 pts

1.1 Problem 1.a 15 / 15

✓ **- 0 pts** Correct

- 3 pts no merging cycles

- 5 pts no decomposition

- 8 pts algorithm runtime not correct

- 10 pts wrong answer but show efforts

- 15 pts no answer

1.2 Problem 1.b 10 / 10

✓ **- 0 pts** Correct

- 3 pts no merging cycles

- 3 pts missing proof

- 4 pts missing algorithm, or missing step-by-step explanation for algorithm

- 8 pts wrong answer but show efforts

- 10 pts no answer

QUESTION 2

2 Problem 2 25 / 25

✓ **- 0 pts** Correct

- 5 pts fail to check if candidate is celebrity

- 10 pts algorithm runtime not correct

- 20 pts wrong answer but show efforts

- 25 pts no answer

QUESTION 3

3 Problem 3 25 / 25

✓ **- 0 pts** Correct

- 8 pts fail to compute max height = max subtree height + second max subtree height

- 8 pts fail to satisfy O(N) complexity

- 3 pts lack step to step details of the algorithm

- 3 pts lack complexity analysis

- 3 pts did not prove correctness

QUESTION 4

4 Problem 4 25 / 25

✓ **- 0 pts** Correct

- 5 pts fail to justify answer, or need proof of correctness

- 25 pts no answer found

- 3 pts fail to show step to step description of the algorithm

- 5 pts does not use recursive call

- 5 pts Click here to replace this description.

- 2 pts Fail to consider case when n==2

CS180 Homework 2

Due: 8:00pm, 1/24/2019

1. An Eulerian cycle of a graph G is a path which starts and ends on the same vertex and traverses every edge in the graph exactly once.

   (a) We have seen that an undirected connected graph $G = (V, E)$ such that all the vertices have even degrees has an Eulerian cycle. Give an $O(|E|)$ time algorithm to find it.

```
Initialize previousLoop to be empty (can be an empty stack to store vertices).
Initialize currentLoop to be empty (can be an empty stack).
Loop (vertex x) :
Start from vertex x, find a cycle that goes back to x, append the cycle to currentLoop.
If all edges have been visited:
    Append previousLoop to currentLoop. Return currentLoop.
If not all edges have been visited:
    Back trace vertices in currentLoop and find vertex y which connect to unvisited edge. Move all vertices in the back trace process before reaching vertex y to previousLoop. Repeat Loop(vertex y) with vertex y.
```

We can first start from a point x, go through some vertices and edges and form a cycle. We know we can form a cycle because each vertex has even degree, so we can always enter and exit (2 degree) at each vertex, except for the starting point x, where we would form a cycle. Now this cycle does not necessarily contain all edges. So we back trace the vertices this cycle go through, and find the first vertex y on the cycle that connects to certain edges that does not belong to the current cycle. And then we can start from y and visit some unvisited edges, and eventually go back to y and form another cycle. Then we can combine cycle x and cycle y. We keep back tracing and combining cycles until the cycle becomes an Eulerian cycle.

Correctness:

This algorithm depends on the fact that we can always find a cycle that goes back to the point where we started, because each vertex has even degree, so we can always enter and exit (2 degree) at each vertex, except for the starting point, where we would form a cycle. Thus by building cycles and combining them, we will eventually find Eulerian cycle.

Complexity:

In the worst case, we back trace every edge, so that in total E edges are backtraced (moved from stack currentLoop to previousLoop), which is O(E). We also visited every edge, which is in total O(E). So the overall complexity is $O(|E|)+O(|E|)=O(|E|)$

1.1 Problem 1.a 15 / 15

✓ **- 0 pts** Correct

- 3 pts no merging cycles

- 5 pts no decomposition

- 8 pts algorithm runtime not correct

- 10 pts wrong answer but show efforts

- 15 pts no answer

(b)A directed graph is strongly connected if every vertex is reachable from every other vertex. Assume that for every vertex in a directed graph G = (V, E) its in-degree equals its out degree, and G is strongly connected. Prove that G has an Eulerian cycle and give an O(E) time algorithm to find it.

The algorithm is extremely similar to the one described in part a. Except that when we back trace, we find a vertex that connects to an outgoing edge which we haven't visited before. And when we form the cycle, we follow the direction of the edge. To avoid redundancy, please refer to algorithm described in part a.

The proof is also extremely similar to the correctness part in part a, that we can always find a cycle that goes back to the point where we started. Because for each vertex, in-degree equals to out degree, thus once we enter from an edge into an unvisited vertex (degree 1, odd), there must exist an outgoing exit edge (degree 1, odd), so that the total degree is even. (degree 2, even). Also because the graph is strongly connected, all the vertices and edges are reachable. Thus we can keep building cycles that contain unvisited edges and combing the cycles, and eventually form an Eulerian cycle.

1.2 Problem 1.b 10 / 10

✓ **- 0 pts** Correct

- 3 pts no merging cycles

- 3 pts missing proof

- 4 pts missing algorithm, or missing step-by-step explanation for algorithm

- 8 pts wrong answer but show efforts

- 10 pts no answer

2. A celebrity among n persons is someone who is known by everyone but does not know anyone. Equivalently, given a directed graph $G = (V, E)$ with n vertices, a directed edge from v_i to v_j represents person i knows person j, a celebrity vertex is the vertex with no outgoing edge and n 1 incoming edges. In the class, we have seen an $O(n)$ recursive algorithm that finds whether celebrity exists or not and it does returns it. The graph G is represented by an n n adjacency matrix M, which is a $(0, 1)$-matrix such that $M[i, j] = 1$ if and only if there is a directed edge from v_i to v_j. Give an iterative $O(n)$ time algorithm to find the celebrity vertex in G, or output none if no one is.

```
Initialize the value of a to be 1. Initialize the value of b to be 2.

For i from 1 to n-1:            //in total n-1 comparisons

    If  M[a,b]=1:

            a=b;

    Else if M[a,b]=0:

            Do nothing;

    b=b+1;

//now check if a is celebrity

For k to be all values from 1 to n except a:

    If M[a,k]=1 or M[k,a]=0

            Return none

Return a
```

Correctness:

The key of the algorithm depends on the fact there can be at most 1 celebrity in n people. Thus we can always take two people A and B: if A knows B, then A cannot be a celebrity; If A does not know B, then B cannot be a celebrity. Therefore we can compare the n people in group of 2, and eliminate at least 1 person in each comparison, and eventually check the remaining person whether he is celebrity.

Complexity:

For the 2 people comparison part (first part), each iteration does 2 operations, so in total it is 2*O(n) which is O(n) time. The final check iterate through n people and is also O(n). So the total time complexity is bounded O(n).

2 Problem 2 **25 / 25**

✓ - **0 pts** Correct

- **5 pts** fail to check if candidate is celebrity
- **10 pts** algorithm runtime not correct
- **20 pts** wrong answer but show efforts
- **25 pts** no answer

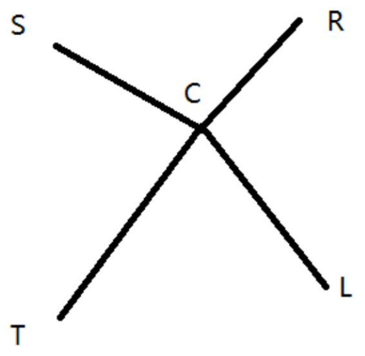
3. Given a undirected tree T, the diameter of a tree is the number of edges in the longest path in the tree. Design an algorithm that find the diameter of the tree in $O(n)$ time where n is number of the nodes in the tree.

```
Initialize an empty queue of nodes, name it firstQueue.

Initialize a map or array that keep tracks of whether the node has been visited.

Initialize a node and name it endNode.

Push a random node into firstQueue.

While queue is not empty:

    Pop the first element x off the queue, mark it as visited, and store it in endNode. Push all the
unvisited nodes that are reachable from x into the queue.

Push endNode into firstQueue.

Initialize count to be 1.

Initialize diameter to be -1.

Loop:

While count is not 0:

    {Pop the first element x off the queue, mark it as visited, and store it in endNode. Push all the
unvisited nodes that are reachable from x into the queue.

    count=count-1}

diameter=diameter+1

count=the size of firstQueue

if count=0:

    return diameter

Go back to Loop
```

Correctness:

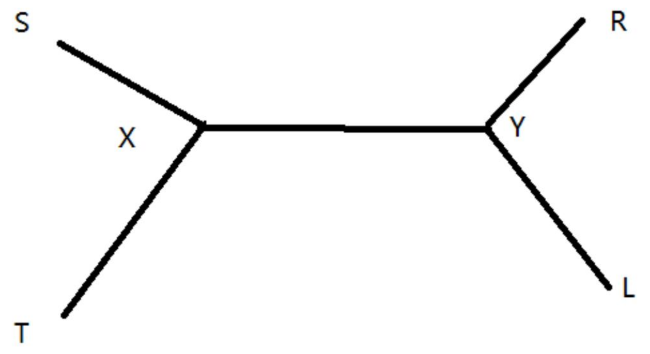
The algorithm first use BFS to find the furthest leaf node L from a randomly given node R, then does another BFS on the founded leaf node R to find diameter. Let the random node be R, the furthest leaf node from R be L. We know that diameter is between two leaf nodes. We assume the two endpoints on diameter to be S and T. We can prove L is on the diameter by analyzing the following situations:

(1) L is not on ST, and RL and ST share a common vertex



Suppose the vertex RL and ST share is C. Because L is furthest leaf node from R, CR+CL >= RC+CT, so CL >= CT, so CL+SC >= CT+SC. If CL+SC > CT+SC, which is the diameter, we reach a contradiction. If CL+SC = CT+SC, then CL+SC is also the diameter, and thus L is on the diameter.

(2) L is not on ST, and RL and ST do not share a common vertex



Because a tree is connected, we can assume ST and RL is connected by X on ST and Y on RL, with XY>0. Because L is furthest leaf node from R, YL >= XY+XT and YL >= YX+ SX. Thus SX +XT < SX+XY+YL. Now that SX+XY+YL is longer than SX and XT, which is the diameter, we reach a contradiction.

(3) L is on ST

Nothing is needed to be proved in this case. L is on the diameter.

Thus we can see that L is always on the longest path, the diameter. And From L we can perform BFS to find the diameter.

Complexity:

In the worst case, the random node we start from is an endpoint on the diameter, then the BFS will cost O(n) complexity. The second BFS always cause O(n) complexity. So the total complexity is O(n)+O(n)=O(n).

3 Problem 3 **25 / 25**

- ✓ **- 0 pts** Correct
- **- 8 pts** fail to compute max height = max subtree height + second max subtree height
- **- 8 pts** fail to satisfy O(N) complexity
- **- 3 pts** lack step to step details of the algorithm
- **- 3 pts** lack complexity analysis
- **- 3 pts** did not prove correctness

4. Let $K_n = (V, E)$ be a complete undirected graph with n vertices (namely, every two vertices are connected), and let n be an even number. A spanning tree of G is a *connected* subgraph of G that contains all vertices in G and no cycles. Design a recursive algorithm that given the graph K_n, *partitions* the set of edges E into $n/2$ distinct subsets $E_1, E_2, ..., E_{n/2}$, such that for every subset E_i, the subgraph $G_i = (V, E_i)$ is a spanning tree of K_n.

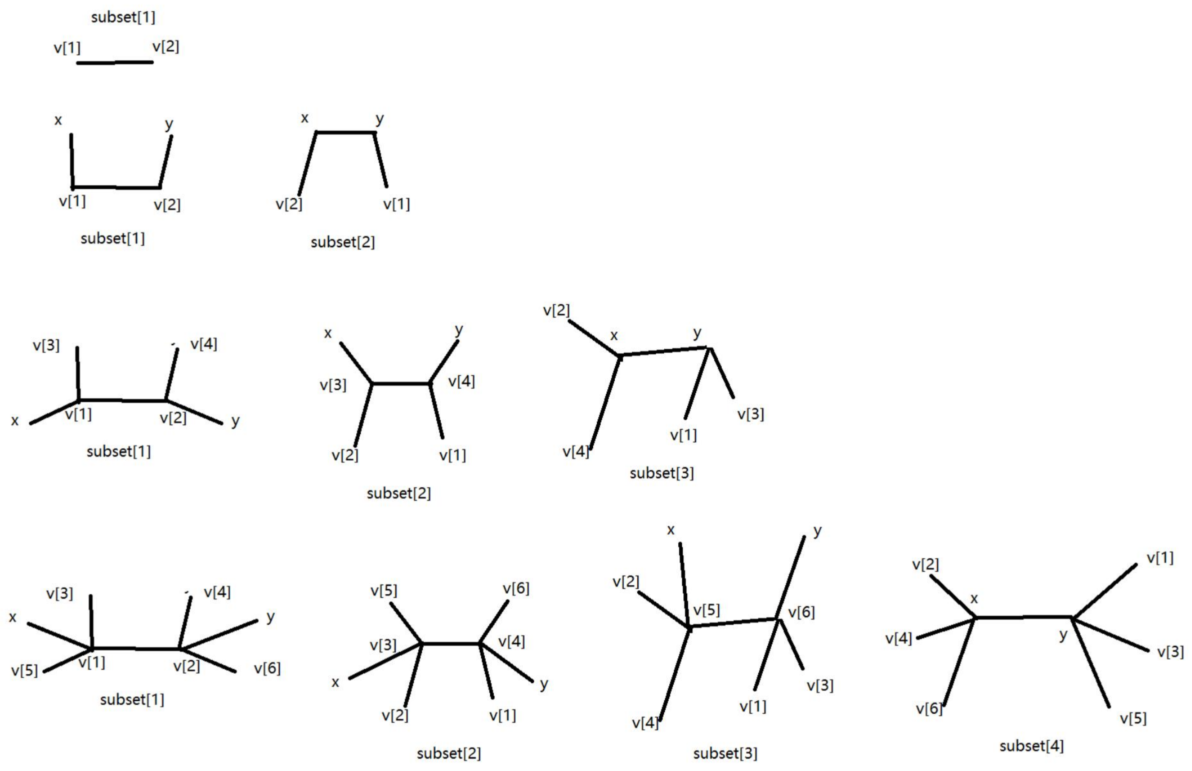
   (Hint: Solve the problem recursively by removing two nodes and their edges in K_n. From the output of recursive call, you get $(n-2)/2$ spanning trees for K_{n-2}. Extend those trees to be spanning trees for K_n and then construct a new spanning tree to complete the job. PS: A collection of sets $S_1, \ldots, S_k$ is a partition of S, if each S_i is a subset of S, no two subsets have a non-empty intersection, and the union of all the subsets is S.)

```
Initialize an empty array of vertices, and name the array A. The length of A is n.
Initialize an array of subsets C, each subset contains several edges. Length of C is n/2.
Push all vertices onto stack B.
//Each entry of A corresponds to a unique vertex.
Function Recursive( int n ){
If A.size() = n
    return C
else
    pop two vertices x and y from stack B.
    initialize i to be 0
    while i< A.size():                    // i takes the value from 0 to A.size()-1
        Add edge E1 between vertex x and vertex A[i], edge E2 between vertex y and vertex
    A[i+1] into C[i/2]
        i=i+2
    initialize j to be 0.
    While j<A.size():
        If j is even:
            Add Edge E3 between vertex y and A[j] into C[A.size()/2]
        Else if j is odd:
            Add Edge E3 between vertex x and A[j] into C[A.size()/2]
        j=j+1
    Add EdgeE4 between x and y into C[A.size()/2]
    Add vertices x and y into A.
Recusive(n);

}
```

Correctness:

This recursive algorithm builds the whole subsets by adding 2 vertices each time. Suppose we have n/2 spanning tree when there are n vertices. Then suppose we have two new vertices x and y. We number the existing subsets from 1 to n/2, and number the existing n vertices from 1 to n. Now, for each existing subset i, we add 2 edges: an edge between vertex x and vertex 2i-1, and an edge between vertex y and vertex 2i. This will generate n/2 spanning tree of n+2 vertices. Finally, we put edge between x and y, edge between vertex 2i-1 and y, and edge between vertex 2i and x together to form a spanning tree of n+2 vertices. This in total generate n/2+1 spanning trees of n+2 vertices.

Illustration diagrams of how the subset grows when n=8:



We know the n/2 subsets of n+2 vertices built upon n/2 subsets of n vertices are spanning trees because we only add two edges, which each has an endpoint of degree 1, and thus cannot be part of a cycle. The new subset we build is also a spanning tree because: (1) it covers all vertices (2) all vertices except x and y has degree 1 (3) the graph is connected and has n-1 edges. Thus we know the algorithm is correct.

Complexity:

Each time we add some edges, the total number of edges added is the total number of edges in the complete graph. Since there are $C\binom{2}{n}=\frac{n*(n-1)}{2}$ edges, the complexity is O(n^2).

4 Problem 4 **25 / 25**

✓ **- 0 pts** Correct

- 5 pts fail to justify answer, or need proof of correctness

- 25 pts no answer found

- 3 pts fail to show step to step description of the algorithm

- 5 pts does not use recursive call

- 5 pts Click here to replace this description.

- 2 pts Fail to consider case when n==2