CS180 HW2

Siddharth Joshi

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

##HW2##

Q1 a)

```
→Let S = an empty stack representing current path
→Let T = an empty list representing the order of vertices
         in the final euler circuit
→S.push (vertex 0) // where vertex 0 is any vertex of the
                    // graph
→current vertex   vertex 0
→while ( S is not empty):
    →if ( there exist unused edges starting at current_vertex)
        →then push current_vertex onto stack S
        →mark the ~~unu~~ an unused edge (v,w) originating
          at v as used.
        → current_vertex = w (the vertex that the unused
            edge from v leads to)
    →else
        → prepend current_vertex to T
        → current_vertex = pop (Stack S)

→Return T
```

Time complexity: The algorithm above has time complexity O(E) as the loop runs while stack S is not empty and every vertex ~~side~~ in graph G must get added to the stack as there are some unused edges that lead to every vertex from a random starting vertex 0 (since G is connected) Moreover, since each vertex leaves the stack only after it has no unused edges originating from it, the while loop runs $\left(\sum_{v \in V} deg(v)\right)/2$ times $= O(E)$ since $E = \sum_{v \in V} \frac{deg(!}{2}$

Correctness: The above algorithm works by starting at a random vertex v and following a path until ~~att~~ it arrives at a vertex ~~x~~ which has no unused edges originating at it, this must necessarily be the starting vertex the first time (as every vertex has even degree ⇒ number of edges ~~us~~ that can be used to exit it = number of edges that can be used to enter it)
At this point, a cycle has been found, the popping of the stack backtracks to find any ~~other~~ vertex that has unused edges, if none exist then by definition all edges have been traversed, else the process repeats with this new vertex effectively inserting a cycle starting/ending at this new vertex into the cycle

b) Since the algorithm used in part a) relies on the fact that ~~the~~ every vertex has even degree to ~~be~~ guarantee that the number of ~~paths~~ edges in to the vertex = # of edges leaving the vertex, the same algorithm can be used on a ~~a~~ strongly connected directed graph where every vertex has ~~even~~ in degree = out degree. Thus, since the algorithm can always construct an eulerian cycle - one must always exist in such a G, and the algorithm to do so has been covered in part a) and has O(E) runtime.

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

Correctness: The above algorithm works by starting at a random vertex v and following a path until ~~att~~ it arrives at a vertex ~~z~~ which has no unused edges originating at it, this must necessarily be the starting vertex the first time (as every vertex has even degree ⇒ number of edges ~~we~~ that can be used to exit it = number of edges that can be used to enter it)
At this point, a cycle has been found, the popping of the stack backtracks to find any ~~other~~ vertex that has unused edges, if none exist then by definition all edges have been traversed, else the process repeats with this new vertex effectively inserting a cycle starting/ending at this new vertex into the cycle

b) Since the algorithm used in part a) relies on the fact that ~~the~~ every vertex has even degree to ~~ge~~ guarantee that the number of ~~paths~~ edges in to the vertex = # of edges leaving the vertex, the same algorithm can be used on a ~~a~~ strongly connected directed graph where every vertex has ~~even~~ in degree = out degree. Thus, since the algorithm can always construct an eulerian cycle - one must always exist in such a G, and the algorithm to do so has been covered in part a) and has O(E) runtime.

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

Q2.

```
FIND FAMOUS ( NxN Adjacency matrix M with rows labelled
                1 . n and cols labelled 1 . n)
→ Let i = 1
→ Let j = n
While ( i ≤ n and j ≥ 0 ):
    • if ( i == j )
        if ( i == n/2 )
            return None if every other row doesn't have a 1 in its ith column
                                                          else return i
        else
            return i
    • else if ( M[i][j] == 0 and M[j][i]   0 )
        increment i ~~a~~ and decrement j
    • else if ( M[i][j] == 0 and M[j][i] == 1 )
        decrement j
    • else if ( M[i][j] == 1 and M[j][i] == 0 )
        increment i
    • else
        increment i and decrement j

Return None
```

Time complexity: The algorithm traverses the n rows both forwards and backwards and hence in the worst case takes n iterations with only one counter being ~~a~~ incremented or decremented ∴ O(n) as all other ops are O(1)

Correctness: The conditions for skipping a ~~row~~ ~~person~~ person A ~~is~~ if ~~and~~ are:

- if person A knows someone else
- if a person B doesn't know person A
    - the algorithm as such traverses every person and narrows down the possible choices to a final answer

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

Q3.

```
→ INT VECTOR DIST ~~⇒~~ INITIALIZE ALL VALUES TO 0
→ QUEU Q = An empty queu
→ PUSH ROOT ONTO Q
→ While (Q is not empty):
    • currentNode = Q.pop()
    • mark currentNode as traversed
    • Push all nodes adjacent to currentNode that have not
      yet been traversed onto the Q
    • Set corresponding values for all untraversed
      neighbour nodes in dist to (dist[currentNode] + 1)
→ Node FarthestNode = Node corresponding to Index of max value of dist
→ Reset all values in dist to 0
→ PUSH ~~FARTHEST~~ FarthestNode onto Q
→ While (Q is not empty):
    • currentNode = Q.pop()
    • mark currentNode as traversed
    • Push all untraversed neighbour nodes of currentNode onto
      Q
    • Set corresponding values for all the aforementioned
      untraversed neighbour nodes of CurrentNode to
      dist[currentNode] + 1
→ Return max value in dist
```

Time complexity: The two while loops essentially execute B.F.S. which is known to be $O(|V|+|E|)$ in general ~~and = $O(|E|)$ for connected graphs as $O(|V|) \leq O(|E|)$~~ Similarly since the statements finding max value in ~~an~~ ~~an~~ vector take O(size of vector) time, in this case they take $O(|V|)$. To conclude the time complexity of

this algorithm can be thought of as O(|V| + |E)
but since the graph is a tree |E| and since a tree
with n vertices has n-1 edges, the time complexity
= O(n + n-1) = O(n)

Correctness: For a tree, the diameter i.e. length of the longest path can correspond to a path contained entirely in one of the subgraphs rooted at the root's children or span across 2 such subgraphs and include the root vertex. In either scenario, finding the vertex farthest from the root and then returning the longest path ~~at the~~ starting at this vertex gives us length of the diameter as in the case that the longest path is contained entirely in a child's subgraph, the initial farthest vertex will be in the said subgraph and hence the longest path from this initial farthest vertex will be to a vertex in this subgraph. If the longest path does cross the root vertex even then the farthest vertex from the root must be at one end of this path and thus the longest path from this vertex will give us the diameter.

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

Q4

```
SPANNING TREE (Graph G):
→ Let trees be a vector ~~of~~ sets of edges
→ Add the set containing the edge (1,2) to trees.
→ Let n = number of vertices in G
→ Let i = 4
→ While ( i ≤ n ):
    • Let edgesA = all edges from vertex i-1 to
      vertex 1 through vertex i-2
    • Let edges B = all edges from vertex i to
      vertex 1 through vertex i-1

    • For each set S in trees:
        - ~~add~~ ~~or~~ remove an edge from edges A ~~and~~ ~~at~~ that
          is incident on a vertex with only 1 edge in this set S, and
          add said edge to S
        - remove an edge from edgesB that is incident on a
          different vertex w only 1 edge in S, and add
          said edge to S

    • Add all remaining edges of edges A and edges B to
      a new set newTree
    • add newTree to trees
    • i += 2

→ Return trees
```

Time complexity: The while loop runs n/2 times and the ~~each~~ for loop runs once for every set in trees at that inner/ point. ∴ Total runtime = $(2 \times 2 + 3 \times 3 + \frac{n}{2} \times \frac{n}{2})$

Ans: $O(n^2)$

Correctness: The initial set in trees that serves as the answer for K_2 is trivially correct (only 1 edge ∴ only 1 way to partition). The inductive step is c i.e. the step building a correct answer for K_n from K_{n-2} is correct as:

- the for loop extends every spanning tree in K_{n-2} to be a valid spanning tree for K_n while never adding any 2 identical edges to any set i.e. it maintains the correctness of the partition
- The new spanning tree again by the nature of the construction has no edges in common w any other set and also is a valid spanning tree as there are necessarily ^ $\frac{n}{2}$ edges left in each set

($\frac{n}{2}$ in $\underset{\text{edgesA}}{\text{one}}$ and $\frac{n}{2}-1$ in $\underset{\text{edgesB}}{\text{one}}$) ∴ the last set ^ also has $n-1$ edges that do not belong to any other set and do not create any cycles ⇒ imply they do form a spanning tree.

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