homework 1

Name: Yifan Zhang

University ID: yz113

Problem 1

> Please prove the following claim: When applying the algorithm we learned form the class to solve stable matching problem, if men propose, every woman gets her worst valid partner (according to her own preference).

Proof:

First, according to stable matching algorithm, every woman will get better and better partner, because a woman will build a connection to a man, only if when she prefer the new applicant rather than her current partner or she don't have a partner yet).

Second, according to stable matching algorithm, every man will get his best valid partner, as we proved in class.

Suppose that in stable matching S', a woman X did not get her worst valid partner in the end and finally her partner is A, and there exist another stable matching S, in which matching, X is matched with man B, whom X likes less than A. In matching S, A's partner is Y.

- Case 1. A prefers X to Y:

  In this case, A-X is an unstable matching in S.

- Case 2. A prefers Y to X:

  In this case, A will not choose X in S'. Because men always choose their best valid partner.

Here is the contradiction. As a result, there will not exist a stable matching, that a woman doesn't get her worst valid partner. So if men propose, every woman gets her worst valid partner.

Problem 2

> A polygon is convex if all of its internal angles are less than 180 degree (and none of the edges cross each other). Figure 1 shows an example.
> We represent a convex polygon as an array V [1, . . . , n] where each element of the array represents a vertex of the polygon in the form of a coordinate pair (x, y). We are told vertex V [1] is the vertex with the minimum x coordinate and that the vertices V [1...n] are ordered

> counterclockwise, as in the figure. You may also assume that the x coordinates of the vertices are all distinct, as are the y coordinates of the vertices.
> Give an algorithm to find the vertex with the maximum y coordinate in O(log n) time.

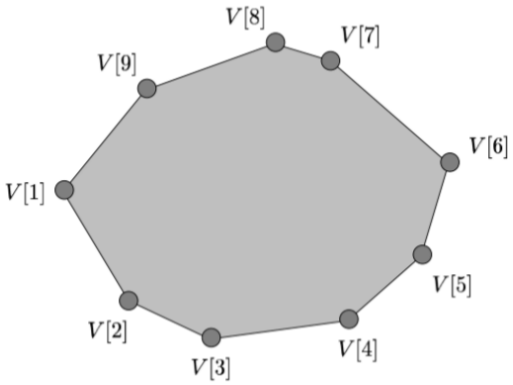

Figure 1: An example of a convex polygon represented by the array $V[1..9]$. $V[1]$ is the vertex with the minimum x-coordinate, and $V[1..9]$ are ordered counterclockwise.

Analyze:

There are 4 states of (x, y) in V:

1. x increases, y decreases
2. x increases, y increases
3. x decreases, y increases
4. x decreases, y decreases

These 4 states will occur in order from V[1] to V[n-1], because the vertices V [1...n] are ordered counterclockwise. In some cases, *some state(s) may not occur, but the order of states will remain.* There will be at least one state.

The maximum y's location will only have 3 possible options:

1. V[0].y
2. v[n-1].y
3. The end of state 3(or state 2, if state 3 is not occur). Meanwhile, it is also the start of state 4.

If the maximum y is in option 3, then state 4 and one of state 2, state3 must be exist, because anyhow, y have to increase to the maximum y, and decrease to avoid v[n-1].y be the maximum y.

It'll be easy to launch a O(log n) algorithm now:

Algorithm:

Input: V[0...n-1]

Output: i (V[i].y is the maximum y in V)

```
# first consider if v[0].y is the maximum y in V
if (v[0].y > v[1].y) and (v[0].y > v[n-1].y):
    return 0
# then consider if v[n-1].y is the maximum y in V
if (v[n-1].y > v[0].y) and (v[n-1].y > v[n-2].y):
    return n-1
# at last consider if maximum y in the middle
```

```
l, r = 0, (n-1)
while (l <= r):
    # the middle point of left edge and right edge
    m = l + (r-1)//2
    # if m is in state 4, move right edge to m
    if (v[m+1].y < v[m].y) and (v[m+1].x < v[m].x):
        # finally, the maximum y will be exposed here
        if(v[m].y > v[m-1].y):
            return m
        r = m - 1
    # else, move left edge to m
    else:
        l = m + 1
```

Problem 3

> You are given a binary tree T = (V, E), along with a designated root node r ∈ V . You wish to preprocess the tree so that queries of the form "is u an ancestor of v?" can be answered in O(1) time. The preprocessing itself should take O(n) (linear in terms of the number of vertices n) time. Please describe and analyze your preprocessing and query algorithm.

Algorithm description

preprocessing algorithm

1. Do a Pre-order depth first search, and label every node their order number as v.nlr. The root node starts from 0.
2. Do a post-order depth first search, and label every node their order number as v.lrn. The root node ends with n-1 (totally n nodes)

query algorithm

1. If u.nlr < v.nlr and u.lrn > v.lrn, then u is v's ancestor

Algorithm analyzation

- preprocess time complexity

  Pre order traversal's and post order traversal's time complexity are both O(n). Overall, the preprocessing algorithm's complexity is O(n)

- query time complexity

  There are only 2 compare instructions in this step. It's easy to see that the query algorithm's time complexity is O(1).

Problem 4

> Judge Jill are dealing with tons of complains, each complaint containing exactly two names: that of the person who filled it and that of the person he or she is complaining about. Jill would like an automated approach to deal with such large amount of complaints.
> She decides to try to label each person involved in the complaint as either good or evil. She only needs the labeling scheme to be consistent, not necessarily correct. A labeling is consistent if every complaint labels one person as good and the other person as evil, and no person gets labeled both as good and evil in different complaints.
> a. Please model this problem as a graph model and propose an efficient algorithm to consistently label all the names as good or evil, or to decide that no such labeling exists. Your algorithm should run in O(m + n) time (linear in terms of the number of vertices n and

> edges m).
> b. Later, Judge Jill wants more than a consistent labeling. She will interview some people to figure out his or her true label, as she can always determine whether a person is good or evil by interviewing him or her. Assuming that there exits a correct labeling such that one person in every complaint is either good or evil, what is the minimum number of people she needs to interview to correctly classify all the people named in the complaints?

- a.

  Description

  This problem can be modeled as "Check whether a given graph is Bipartite or not". We can run a BFS on this graph, and label every node as good if their parent node is evil. otherwise the node will be marked as evil.

  If when the algorithm goes to node v, and we found that the v's child node's label is already marked and it is the same as v's label, then the algorithm return false means that no such labeling exists. Otherwise, this algorithm will consistently label all the nodes' names as good or evil.

  Analyzation

  This is basically a BFS algorithm, and the time complexity is O(m + n)

- b.

  We can use Hungary algorithm to handle this problem. Hungary algorithm can tell us the size of the maximum matching, and it is equal to the size of minimum vertex cover, which is the minimum number of people she needs to interview to correctly classify all the people named in the complaints.

  The time complexity is O(m*n)

Problem 5

> Given an undirected graph, please give an O(n) time (linear in terms of the number of vertices n) algorithm to detect whether the given graph contains a cycle.

Algorithm description

1. generate a global empty hash table H
2. start from arbitrary vertices v, put v in H.
3. for every node vi that connected to v, if vi is already in table H, return true(There is a cycle in given graph) . Otherwise, delete the edge between vi and v. Then, recursively run this step, use vi as input node, until finally found a cycle in the graph, or searched every node in given graph. This step can be recursively up to n times.

Problem 6

> Suppose that an n-node undirected graph G = (V, E) contains two nodes s and t such that the distance between s and t is strictly greater than n/2.
> a. Show that there must exist some node v, not equal to either s or t, such that deleting v from G destroys all s-t paths. (In other words, the graph obtained from G by deleting v contains no path from s to t.)
> b. Give an algorithm to find such a node v with O(m + n) running time (linear in terms of the number of vertices n and edges m).

- a.

Suppose there are arbitrarily two paths A and B from s to t that contain completely different node. Namely, every node on path A and path B are different, except s and t.

The distance between s and t is strictly greater than n/2, which means that there will be strictly more than n different nodes on path A and path B. However there are only n nodes in graph G. Proved by contradiction.

As a result, there must exist a node v, not equal to either s or t, such that deleting v from G will destroys every arbitrary path from s to t.

- b.

  Using Tarjan on the whole graph, and check the graph again, for each node u, if low[v]>=dfn[u] for some v, that v is the son of u, then u is a such node. We just need to check each node once and check all the nodes connected to it that is not visited, by DFS or BFS. The time complexity is O(n+m).

Problem 7

> The police department in the city of the Wonder Land has made every street in Wonder Land one-way. Despite widespread complaints from confused motorists, the mayor claims that it is possible to legally drive from any intersection to any other intersection.
> a. The city needs to either verify or refute the mayor's claim. Formalize this problem in terms of graphs, and then describe and analyze an algorithm to solve it.
> b. After running your algorithm from part (a), the mayor reluctantly admits that she was misinformed. Call an intersection x good if, for any intersection y that one can legally reach from x, it is possible to legally drive from y back to x. Now the mayor claims that over 95% of the intersections in Wonder Land are good. Describe and analyze an efficient algorithm to verify or refute her claim.
> For full credit, both algorithms should run in O(m + n) time (linear in terms of the number of vertices n and edges m).

- a

  Description

  Define every intersection as a node and every street as a direct edge in a directed graph G(V, E).

  1. Pick up a arbitrary node v.
  2. Run a DFS, use v as root node. If DFS can't find every node, then algorithm ends and the mayor's claim is wrong.
  3. Transpose the graph G(reverse every direct edge), and run a DFS on node v again. If DFS can't find every node, then algorithm ends and the mayor's claim is wrong. Otherwise, the mayor's claim is right.

  Analyze

  - time complex

    2 DFS in the algorithm. Every DFS's time complexity is O(m+n), and overall's time complexity is O(m+n)

  - correctness

    Pick up an arbitrary node v, if there exist an path from v to any node, and meanwhile, exist an path from any node to v. Then every two node in this graph is connected. Then the mayor's claim is right. Otherwise is wrong.

- b

Using Tarjan to get the connective components, adding a array vis[i] to record the component that each node when the node is popped out of the stack and set vis[i] as the id of the current round, the id of round is added by 1 when execute the Tarjan algorithm each time then we can check the vis array after find all the SCCs and get the amount of the nodes in this SCC and check if it is bigger than 95% of amount of nodes.

The time complexity is O(n+m).