Homework 1

Due date: 1/13/2021 right before class

Problem 1

We have seen in class a centralized algorithm to find stable matching given n men, and n women, and for each their preference.
Consider a distributed version. Think of the men and women as real people. They communicate by sending emails to each other. The elementary operations are:

1. Man or Women sending a message (The message is now in transit. Men sends proposals Woman send rejections).

2. A message in transit delivered.

3. Man or Woman reads a delivered message it has not seen before and responds accordingly.

The men and the women behave as in the algorithm. I.e. a woman does not respond to a proposal she accepts, and send rejects to any proposal of lower rank than the proposal she currently did not respond to. A man upon receiving rejection goes to the next woman in rank order and send her a message.

Will the process enter *quiescence*? no message on transit and no man or woman will generate any message any more?

Then the unresponsive proposals constitute the match. In distributed setting we now need a process called termination to detect that we entered this quiescent system state, but that will not an issue will will deal with in this problem.

Solution 1

Yes the system will enter quiescence. Assuming that eventually all messages that are queued will be sent, all men have a limited amount of women they can propose to, and women then have a limited number of men they will reject until they are done responding. A man has sending a message has two options once he sends a message:

1. Waiting for a response.

2. Received a response and either send a new message or is done.

If he sends a new message the same state repeats itself, until he is stuck waiting or is done. Therefore he will eventually be silent.

A woman has two options:

1. She accepted a proposal and is then sending rejections.

2. She did not receive a proposal yet.

If she sends rejections, we know there will be a final amount of them based on the behavior of men. If she did not receive anything she is silent and contributes no action to the system.

Another way to look at this is that by contradiction, if the process does not enter quiescence, there must be at least one message in flight at any time. In turn that means that there always exists a man that proposes to a women or a women that is rejecting a man. Since the amount of messages men initially send are finite, this introduces either a cyclic process or a new player to the system, both contradict the way that the system works.

Problem 2

Stating our matching problem as a bipartite graph, Hal's theorem states:

"Let G be a finite bipartite graph with bipartite sets X and Y (i.e. $G := (X + Y, E)$). An X-perfect matching is a matching which covers every vertex in X.

For a subset W of X, let $N_G(W)$ denote the neighborhood of W in G, i.e. the set of all vertices in Y adjacent to some element of W. The marriage theorem in this formulation states that there is an X-perfect matching if and only if for every subset W of X: $|W| \leq |N_G(W)|$. In other words: every subset W of X has sufficiently many adjacent vertices in Y."

We add to our stable match problem the ability to mark an individual of the opposite sex as undesirable, i.e, an individual marked as undesirable by another person could not be considered as a match for that person.

Given these two pieces of information, explain under what conditions will a stable match still exist.

Solution 2

Representing our graph as a bipartite graph $G := (X + Y, E)$ we can see that an undesirable state will mean that no edge would be in the bipartite representation. We can now use Hal's theorem and check if a perfect matching exists in our bipartite graph, since a perfect match is a necessary condition for a stable match, i.e: we need at least each man proposing to at least one woman and every woman to receive at least one proposal. In other words, no one is marked undesirable by everyone. From Gale Shapely we have proved that when we have two disjoint vertex sets of equal size that it is always possible to find a stable matching.

Problem 3

Given a number n, we know how to describe it as a set of binary operations. We add 1 n times and perform a number of bit operations in the process. In lecture we bounded this operation from above by $O(Log(n))$ To arrive at total uper bound of $O(n\ Log(n))$. But this may be in the popular word of these days a Fake upper-bound, i.e. there exists a better upper-bound.
Find a lower upper bound to describe this process. Notice that when the current number to which we have 1 now is even then numbers of operations needed for addition is 1.

Solution 3

Notice that for each element in the array representing our binary number (call it B) each element $B[i]$ will be accessed exactly $\frac{n}{2^i}$ times. So we can sum the total amount of operations $\sum_0^{log(n)} \frac{n}{2^i} = n(2 - 2^{-log(n)})$ which in turn translates to $O(n)$ complexity.

Problem 4

Write a recursive algorithm given the number n to find its binary representation. A recursive algorithm checks whether the number is 0. If it is it stops and does nothing (the initial array of 0s is the binary rep of n). Else it calls for the binary rep of some number n' smaller than n and then from the binary rep of n' gets the binary rep of n.

In class we essentially have seen a choice of $n' = n - 1$. What if we chose $n' = n \div 2$? How would you recover in few steps the binary rep of n from that of $n \div 2$?

This now will be a recursive algorithm. A recursive algorithm gives "control" to the computer. Now we want to translate it into an iterative algorithm, to get the algorithm Aryaman started to tell us in class. Write the recursive and the iterative algorithms and analyse the complexity of the latter.

Solution 4

```

arr = [0,0,.....,0] (floor(logn)+1 bits)
i=0
```

```
DectoBin(n, i, arr):
    if n=0: return arr
    arr[i]=n%2
    return DectoBin(n//2, i+1, arr)
```

Listing 1: Recursive algo

```
k=0
arr=[0,0,....,0]
DectoBin(n):
    while n>0:
        A[k] = n%2
        n//=2
        k+=1
    return arr
```

Listing 2: Iterative algo

We devide our problem space in half every loop, and have two constant operations each loop, therefore we will make O(Log(n)) operations.

Fun Problem

If you want, I'll look at any respose that think they have the answer. Intangible credit. You do not *need* to do this problem. It not really part of the HW1!

Suppose you have an initial acyclic directed graph (”no police station” and consider the distributed process of making sink into a source. The process will continue forever. Since the graph is finite we will be able to rearrange timing such that the process enters a period. Can you say something on this period. If interested look up the internet on PhD Thesis or a book by Valmir Barbosa. The acces the original paper of the “police cars” look up Eli Gafni, Dimitri P. Bertsekas: Distributed Algorithms for Generating Loop-Free Routes in Networks with Frequently Changing Topology. IEEE Trans. Commun. 29(1): 11-18 (1981)