Assignment 4
CSC425
Zhaocheng Li
V00832770

Question 1:
Recall:
An augment path is an alternating path where the first and last node is unmatched. Thus a path switches in M and not in M while the endpoints are not in M.

Recall (Shortest Augmenting Path Algorithm):
Assuming there is perfect matching, when the new client c arrives and it connects to certain servers, it chooses the shortest augmenting path P if possible, and augment down path P by flipping all the edges, beginning at node c to a server.
Therefore after n clients have arrived, we will have a perfect matching of size n.

When such a client c is not matched at it arrives, meaning that, meaning that nothing changed in the bipartite graph except there are few more unmatched edges connected to some servers.

So the only hope for c to get into the matched system is being flipped when it is inside the shortest augmenting path of one following incoming client. But it must request c to two servers using matched edges but as an unmatched note given the fact above. It is simply impossible.

So as the clients keep coming, it is impossible for a node that is not matched to be matched again.

Question 2:
Suppose we have a stream of words incoming and the total size is n.
Let A be an array of size n, and the index i corresponds to the word i.
Initial configuration is that the array A is 0's for all entries.

At each point in time, the item i can either be inserted or deleted from A, we increment A[i] by 1 for insertion, and decrement A[i] by 1 for deletion. In this case, at any time, A[i] equals the number of copies of i in the data, or in other

words, the frequency of word i. Thus we can also assume that A[i] is never less than 0.
However we only have O(logn) space to work with. Then we require logarithmic in the space to represent A explicitly. We think of entries of A as words and count the space in terms of number of words.

We will estimate A[i] at any given time

Assuming with the probability of success 1-alpha, the error for which our estimation is wrong is within the factor of beta,

We will use hash functions in the algorithm.
Pick log(1/alpha) hash functions hj: [n] → [e/beta], which is chosen uniformly from a family of pairwise hash functions. Then we think of hj(i) as a bucket for i corresponding to the jth hash functions. And we set counter for each bucket. All buckets and counters are set 0 initially.

Then every time when a word i is inserted, we increment the counter by 1 for the according bucket hj(i) for all hash function (for all j).

In this case the storage is need with in O(logn) but I can not figure out a way to calculate the probability.

Question 3.
PART a: for the algorithm
By the hint, the input should consists of two pairs for each edge {u,v} in G of the form $(u,\ (u,v))$ and $(v,\ (u,v))$. where $u,v \in V$

INPUT: in the form of $(u,\ (u,v))$, one for each endpoint in each edge, all edges in E.

Phase 1: Mapper

1. PARTITION input pairs into k random equal size sets, $V_1 \ldots V_i \ldots$, given $|V| = N$, so $|V_i| = N/k$.
2. EMIT: Among all the qual size sets, Let E_{ij} be the subset of edges in E, with both endpoints in $V_i \cup V_j$, for each pair V_i and V_j. Let $G_{i,j} = (V_i \cup V_j, E_{ij})$. Send mapper to each input tuple for each pair

(associated with two endpoints of a edge, such as {u,v}), if there is any such pair of input. Emit a new pair $(G_{ij},(u,v))$ by {key= G_{ij} ,value= (u,v) } .

3. OUTPUT a new set of $(G_{ij},(u,v))$ tuples, and using the key, send the tuple to reducer with the same key.

Phase 2: Reducer, each reducer is responsible for all pair with the same key= G_{ij}

For each reducer, we compute the MST of G_{ij} .

Consider a tree M_{ij} , let H be the graph consisting of all of the edges present in some M_{ij} and $H = (V, \cup_{ij} M_{ij})$.

Finally, compute M, the minimum spanning tree of H by output (M_{ij} , (u,v)).

Then M is the MST of G.

PART b:

in phase 1, we know each E_{ij} fits on one machine with $O(n^{1-\varepsilon})$ size if we partition the vertices into W, groups by degree, e.g., w_0 will be vertices with degree one. Then we bound the total length of edges

$$|E_{ij}| \le \sum_{v \in V_i} deg(v) + \sum_{v \in V_j} deg(v),$$

then for $i = 1, 2, ..., \lceil log\ n \rceil$, let $Wi = \{v |\ 2^{i-1} < deg(v) \le 2^i\}$.

Then if $|Wi| < 2N^{c/2}\ log\ N.$ And if all endpoints of these edges are in V_j

We have space on a machine for all of the edges incident to Wi ,

Then maximum contribution of Wi to B machine is $2N^{c/2} log\ n\ *\ N = 2N^{1+2/c} log\ n$,

So the total data must be $O(N^{1+c/2} log\ n)$ per machine,

and for each tuple (ket,value), the size must be $O(log\ n)$. Otherwise, we cannot partition them in to w_i groups, where i =1,2,... $\lceil log\ n \rceil$.

Question 4:

Based on the Ben-Or paper, we describe the terms "traitors", or "bad generals" we used in Byzantine Agreement problem, as the t numbers of processes, that make a finite number of steps as any message being eventually delivered if their receiving process make an infinite number of steps. Where "step" means the process takes its one configuration to one another, then makes correspondence and computation based on what it has received.

First of all, for a totally N processes including t "bad processes", to guarantee that there is a chance we can reach an agreement, the lower bound should be

$(N-t) > (N+t)/2$

And it leads to

$T < N/3$.

In order to explain how Ben-Or algorithm would fail on case $N/5 \pounds t < N/4$, we can simply give a counterexample such that there exists a probability that the agreement will never be found, i.e., the algorithm will be in an infinite loop. From lemmas of Ben-Or algorithm, if all the good processes have the same v initially, the agreement will be found in one round; If some processes have an agreement with a value v at round m, then the rest processes will have the agreement with the same value of v in the next round.

The lemma from professor note also states that if some processes reach the algorithm 3a, them every process is in either 3a, or 3b.

Therefore, for a case which the agreement can not be found, we must make sure at any round, any good process can not find an agreement.

For example, when t=1, and n=5t=5, thus there are 4 "good processes" and one "bad process", for each process, we need to receive more than 3 common value of v to be passes out again. Otherwise we will randomized its value with probability ½ for value 1 or 0.

Therefore we can see that if the initial configuration for the good processes is 2 1's and 2 0's, and in every round, all processes cannot receive more than (N+t)/2 same vote, and receive either 0 or 1 in step 3c restricted following the probability distribution. In this way, we will never have an agreement iteratively in this case,