CIS502 HW1

Zi Yan

September 18, 2011

Problem 1

In this case, G-S algorithm still can work fine with indifferent preference. The only difference needed to mention is when an engaged woman w receives a proposal from m' which is on the same position with currently engaged man m, w will not switch, because m' do not have a higher position than m.

Therefore, the facts in original stable matching case still hold, and the runtime of the algorithm is still $O(n^2)$

(a) **Solution** Facts (1.1) (1.2), (1.3), (1.4), (1.5), and (1.6) still hold, therefore, G-S algorithm can still produce a perfect matching without any strong instability.

(b) **Solution** For the first part of the condition of a weak instability, it is the same as a strong instability, so we only need to prove that the second part, namely if m prefers w to w', w is indifferent between m and m', or if w prefers m to m', m is indifferent between w and w', will not hold when G-S algorithm is applied.

Since facts (1.1) (1.2), (1.3), (1.4), and (1.5) still hold, we only need to prove that the produced set S from G-S algorithm has no weak instability. We do it by contradiction, assuming there is a weak instability. Based on the first condition mentioned above (the proof for second condition can be adapted from the first one), m's last proposal is w'. So did m propose to w before? If m did not, then w' is higher than w in the preference list of m, but it contradicts the assumption. If m did, then m was rejected by w, because of some better m''. m' is w's final partner, so it means either $m'' = m'$ or w prefers m' to m''. Both will

contradict the assumption that w is indifferent between m and m'. In sum, there will be no weak instability in the result of G-S algorithm.

Problem 2

Algorithm: For each Input, create a preference list such that each Input prefers an upstream junction over a downstream one. Each Output prefers a downstream junction over a upstream junction. Given these preferences, there exists a stable matching, which we can find in time $O(n^2)$ using the Gale- Shapely algorithm. A valid switching of the data streams can always be found.

Proof. We need to prove that no Input i, switching onto a downstream Output from j, is connected with Output j onto which Input k switches, so the junction box of i and j is downstream from the one of i and k on Output j. Suppose not. Suppose Input i switches onto Output r. The Input i prefers Output j to Output r (since Output j is upstream from Output r on Input i) and Output j prefers Input i to Input k (since Input i is on downstream from Input k on Output j). This contradicts the fact that we had a stable matching. □

Running time: Constructing the preference lists takes $O(n^2)$ time, and G-S algorithm runs in time $O(n^2)$. Thus the overall running time is $O(n^2)$.

Problem 3

Proof. Suppose for a certain preference list, w has the final partner $\overline{m}$, and prefers m to m'. After w changed the preference list falsely, showing that she prefers m' to m, she will get a better m'', which ranks higher than $\overline{m}$, m, and m'.

There are three cases:

(1) w prefers $\overline{m}$ to m in the original preference list. In this situation, once w is proposed by $\overline{m}$, she will always switch to $\overline{m}$, despite the ranks of m and m'. So $m'' = \overline{m}$, which contradicts the assumption.

(2) $\overline{m} = m$. So once w falsely claims she prefers m' to m, either she will switch from m to m', or she will reject m because of m'. Finally, $m'' = m'$ is lower than $\overline{m}$, which contradicts the assumption.

(3) $\overline{m}$ is lower in rank than either m or both m and m'. This contradicts the assumption.

In sum, w cannot find a better m'' by falsely claiming that she prefers m' to m. □

Problem 4

For this *highest safe rung* problem, it needs a trade-off between binary search and linear search. So my strategy will combine binary search and linear search, which builds a m-ary tree. In this case, the number of jars stands for the maximum node search times, namely the tree depth, and the total dropping times will be regarded as the tree depth plus the total search number in all visited nodes.

Let m be the number of rungs we will step over between two dropping points, namely the number of keys in a node. Therefore, the number of jars k, namely the tree depth, will be $\log_m n$ and the total dropping times will be $O(k + k \times m)$

(a) when $k = 2$, we have $\log_m n = 2$, so $m = \sqrt{n}$.

The **strategy** will be that we will first drop the jar every $\sqrt{n}$ rungs, when we break the first jar at $(i \times \sqrt{n})$th rung, then we will start to drop the second one from $((i-1) \times \sqrt{n})$th rung, and go higher one rung by one rung until the jar is broken. Finally, we can find the *highest safe rung*. Therefore, $f(n) = O(2 + 2 \times \sqrt{n}) = O(\sqrt{n})$.

Because we already have the runtime $f(n) = O(\sqrt{n})$ from above, $f(n)$ is known to grow slower than linearly.

(b) when $k > 2$, we have $log_m n = k$, so $m = n^{\frac{1}{k}}$.

The **strategy** will be that with 1st jar, we drop it from the bottom every $n^{\frac{k-1}{k}}$, ..., with ith jar, we drop it from $((j-1) \times n^{\frac{k-i+1}{k}})$ to $(j \times n^{\frac{k-i+1}{k}})$ every $n^{\frac{k-i}{k}}$ rungs (if we broke the $(i-1)$th jar at $(j \times n^{\frac{k-i+1}{k}})$th rung), ..., with kth jar, we drop it from $((j-1) \times n^{\frac{1}{k}})$ to $(j \times n^{\frac{1}{k}})$ every rungs (if we broke the $(k-1)$th jar at $(j \times n^{\frac{1}{k}})$th rung). Therefore, $f_k(n) = O(k + k \times n^{\frac{1}{k}}) = O(n^{\frac{1}{k}})$, assuming $k < n^{\frac{1}{k}}$.

Because we already have the runtime $f_k(n) = O(n^{\frac{1}{k}})$ from above, then $\lim_{n\to\infty} f_k(n)/f_{k-1}(n) = \lim_{n\to\infty} n^{\frac{1}{k}}/n^{\frac{1}{k-1}} = \lim_{n\to\infty} n^{-\frac{1}{k\times(k-1)}} = 0$