CMSC 27200 Assigment 1

Rachel Hwang

February 25, 2014

- **Exercise 1.1**

  False. Suppose we have a sets of men $\{m, m'\}$ and women $\{w, w'\}$ with matching preferences as follows

  $$\begin{array}{ll} m: w, w' & w: m', m \\ m': w', w & w': m, m' \end{array}$$

  Using the Stable Matching Algorithm, this results in the set $S = \{(m, w), (m', w')\}$. There is *no* pair (m_i, w_i) such that m_i is ranked first on the preference list of w_i and vice versa.

- **Exercise 1.3**

  Counterexample: Let the number of schedule slots $n = 2$. Network A has two shows, ratings {4, 2} and Network B has two shows, ratings {3, 1}. The possible schedule pairs (A, B) are

  $$\begin{array}{ll} (4,3)\ A\ win & (4,1)\ A\ win \\ (2,1)\ A\ win & (2,3)\ B\ win \end{array}$$

  Network A will always prefer the first schedule and Network B will always prefer the second. Thus, given a set of shows and ratings, there is not always a stable set of schedules.

- **Exercise 1.7**

  This problem is analogous to the stable matching problem. In this case, the men are the input wires, the women are the output wires and the stable output set is a set of pairs (I, O). Note that there is precisely one sequence of junction boxes, one path, that joins the source of a given input wire to the terminus of a given output wire. Each input or output wire has a ranked list of preferences about which coresponding wire it would like to pair with.

  The preferences of the wires are determined by the order of junction boxes on them. An input wire most prefers the output wire accessible by the junction box most upstream along itself: the more upstream the junction box, the more preferable the output wire. The preferences of the output wires are determined by how close to its terminus (how far downstream) each input wire joins with it; output wires prefer the inputs from the most downstream junction boxes. Because the configuration of junction boxes on a wire is a strict ordering, there are no ties in any wire's preferences.

  Given this definition of preferences, we can see that having two datastreams cross (two paths using the same junction box) is equivalent to instablity in a stable-matching. Given any potential input/output wire pair (creating a datastream path), the path travels along its input wire before picking one junction box, hence one output wire, to claim and continues along that output wire. Since each input wire joins with any given output wire at a unique junction box, paths will never cross at a junction box when both paths are traveling along input wires. Similarly, since each path must exit through a unique output wire, paths will never cross when both paths are traveling along output wires. This means that in any attempt at a valid switching, paths only cross when one path is traveling downstream on an output wire, already commited to the output wire that it is on, and the other intersecting path is still traveling along its input wire, just passing through the junction box on that output wire. Using the input and output wire preferences as stated above, this is equivalent to instability because such a crossing means that wire pairings could be changed in order to benefit both wires involved. Rather than just passing across, the input wire should join with that output wire. This switch results in a more preferable matching for both wires. Thus, crossed

datastreams always represent an instability and so a stable matching of input and output wires will be free of crossings.

Since we have shown that this problem is equivalent to the stable-matching problem, a valid switching can be found using the Gale-Shapley algorithm. We have previously proved that the algorithm always produces a stable matching, therefore we can always find a valid switching with no crossing.

- **Exercise 1.8**

True. A woman can get a preferable partner by lying about her preferences. Suppose we have a sets of men $\{m, m', m''\}$ and women $\{w, w', w''\}$ with truthful matching preferences as follows

$$\begin{aligned} m &: w, w'', w' & [w &: m'', m', m] \\ m' &: w, w', w'' & w' &: m, m', m'' \\ m'' &: w', w, w'' & w'' &: m', m'', m \end{aligned}$$

Using the Stable Matching Algorithm, this results in the set $S = \{(m, w''), (m', w), (m'', w')\}$. However, if w lies and ranks her preferences as $[m'', m, m']$, then the resulting stable match will be $S = \{(m, w''), (m', w'), (m'', w)\}$, giving w her first choice man rather than her second choice.

This happens because by ranking m higher than m', she remains with m and allows w' to be paired with m'. Now that w' is already happily matched, m'' proposes to and is paired with his second choice, w.

- **Exercise 3.9**

Proof: Given a graph with n nodes, among them nodes s and t such that the distance between them is greater than $n/2$. There exists node $v \neq (s \text{ or } t)$, such that removing v will destroy all paths between s and t.

Imagine that each node is labeled with its distance from s, i.e. all the direct neighbors of s are labeled 1 and so on. Given that the distance d from s to t is greater than $n/2$, there must be at least one label value k, where $0 < k < d$, such that only one node bears that label (only one node is precisely that distance from s). As proof, assume this is not the case. If there are at least two nodes per label value, then there are more than $(d-1) \cdot 2 = 2d - 2$ nodes, plus s and t, which gives a total of $2d > n$ nodes. This creates a contradition, so it must be the case that there is one unique label value k.

This k-labeled node is v. Since it is the only node that distance from s and less than d, we know that all paths between s and t must include the v node. (This is because as one traces a path, the label of the vertex can change by at most 1. Therefore, going from label 0 to d means we must pass through k since $0k < d$). Because all paths include this node, removing it will destroy all paths between s and t. Thus we have shown that v must exist.

The algorithm for finding v is simple: since we know that v is a unique distance from s, we can perform a breadth-first traversal of the graph beginning at s. Each layer in our traversal is one unit of distance further away from s. As proved above, there must be at least one layer which contains only one node. As soon as the algorithm finds such a lone node, that node is a valid v. Breadth-first traversal is described in the textbook and proved to run in O(m+n) time.

- **Consider the brutally slow algorithm for Stable Matching that generates all matchings then tests each whether it is stable. Suppose it takes unit time to generate a matching. Suppose that your computer is so advanced that "unit time" is the time it takes light to traverse the diameter of an electron. What is the largest n such that all possible matchings could be generated, if the time spent is the estimated age of the universe?**

For input size n, the number of total possible matchings is $n!$. So

$max\_matchings =$ (age of the universe)/((diameter of electron)/(speed of light))
$max\_matchings \approx \frac{4.3 \cdot 10^{17}}{(2 \cdot 2.81794033 \cdot 10^{-}15m)/(2.998 \times 10^{8} m/s)} \approx 2.3 \cdot 10^{40}$

$n! < 2.3 \cdot 10^{40}$
$35! < 2.3 \cdot 10^{40} < 36!$
$n = 35$

- **Extra Credit: Exercise 2.8**

  a. It is optimal to evenly distribute the amount of work that each jar does. One jar will be dropped at large intervals. As soon as it breaks, the second jar will start testing from the next rung above the last safe drop height, since the threshold rung must be somewhere between the last two drops of the first jar. If the first jar breaks on the first drop, the second jar begins at the first rung. When the second jar breaks, the previous rung must be the last safe height.

  To divide work evenly, the first jar should take steps of size $n^{1/2}$ (meaning, drop at rungs $n^{1/2}$, $2 \cdot n^{1/2}$, etc.) and the second will take steps of size 1. This ensures that each jar is dropped no more than $n^{1/2}$ times. For example, given 100 rungs, the first jar will be dropped at heights 10, 20, 30, ... Let's say it breaks at height 70. The second jar begins testing at 61, then 62, etc.
  Using this strategy, the maximum number of drops needed, $f(n) = n^{1/2} \cdot 2$, which runs in $O(n^{1/2})$, which satisfies $\lim_{n\to\infty} \frac{f(n)}{n} = 0$.

  b. Again, we want to divide work evenly. We can do so by extending the above strategy. Given k jars, the first jar will be dropped at intervals of $n^{(k-1)/k}$, the second jar will be dropped at $n^{(k-2)/k}$ and so on, up to the last jar, dropped at intervals of n^0, 1.

  We will see by induction, $f_k(n) = k \cdot n^{1/k}$.
  Base case: $f_1(n) = n$

  Now suppose we have k jars. The first jar is dropped at rungs $n^{(k-1)/k}$, $2 \cdot n^{(k-1)/k}$, up to a maximum number of $n^{1/k}$ drops. When it breaks, the threshold must be between the last two drop heights, somewhere in the $n^{(k-1)/k}$ rungs between them. Now we have $k-1$ jars to search $n^{(k-1)/k}$ rungs, which by our induction hypothesis takes no more than $(k-1) \cdot (n^{(k-1)/k})^{1/(k-1)} = (k-1)n^{1/k}$ drops. Along with the first jar, this is a total of no more than $kn^{1/k}$ drops.

  Since $f_{k-1}(n) = (k-1)n^{1/(k-1)}$ and $f_k(n) = kn^{1/k}$, we know that $f_k(n)$ has a smaller exponent than $f_{k-1}n$. Thus $\lim_{n\to\infty} \frac{f_k(n)}{f_{k-1}(n)} = 0$.