CS222 Homework 1

Stable Matching and Algorithm Analysis
Deadline: 2018-9-25 Tuesday 24:00

Exercises for Algorithm Design and Analysis by Li Jiang, 2018 Autumn Semester

1. Decide whether you think the following statement is true or false. If it is true, give a short explanation. If it is false, give a counter example.

   True or false? Consider an instance of the Stable Matching Problem in which there exists a man m and a woman w such that m is ranked first on the preference list of w and w is ranked first on the preference list of m. Then in every stable matching S for this instance, the pair (m, w) belongs to S.

2. For this problem, we will explore the issue of truthfulness in the Stable Matching Problem and specifically in the Gale-Shapley algorithm. The basic question is: Can a man or a woman end up better off by lying about his or her preferences? More concretely, we suppose each participant has a true preference order. Now consider a woman w. Suppose w prefers man m to m', but both m and m' are low on her list of preferences. Can it be the case that by switching the order of m and m' on her list of preferences (i.e., by falsely claiming that she prefers m' to m) and running the algorithm with this false preference list, w will end up with a man m' that she truly prefers to both m and m'? (We can ask the same question for men, but will focus on the case of women for purposes of this question.) Resolve this question by doing one of the following two things:

   (a) Give a proof that, for any set of preference lists, switching the order of a pair on the list cannot improve a woman's partner in the Gale-Shapley algorithm; or

   (b) Give an example of a set of preference lists for which there is a switch that would improve the partner of a woman who switched preferences.

3. Some of your friends are working for, a builder of large communication networks, and they are looking at algorithms for switching in a particular type of input/output crossbar.

   Here is the setup. There are n input wires and n output wires, each directed from a source to a terminus. Each input wire meets each output wire in exactly one distinct point, at a special piece of hardware called a junction box. Points on the wire are naturally ordered in the direction from source to terminus; for two distinct points x and y on the same wire, we say that x is upstream from y if x is closer to the source than y, and otherwise we say x is downstream from y. The order in which one input wire meets the output wires is not necessarily the same as the order in which another input wire meets the output wires. (And similarly for the orders in which output wires meet input wires.) Figure 1.8 gives an example of such a collection of input and output wires.

   Now, here's the switching component of this situation. Each input wire is carrying a distinct data stream, and this data stream must be switched onto one of the output wires. If the stream of Input i is switched onto Output j, at junction box B, then this stream passes through all junction boxes upstream from B on Input i, then through B, then through all junction boxes downstream from B on Output j. It does not matter which input data stream gets switched onto which output wire, but each input data stream must be switched onto a different output wire. Furthermore-and this is the tricky constraint-no two data streams can pass through the same junction box following the switching operation.

   Finally, here's the problem. Show that for any specified pattern in which the input wires and output wires meet each other (each pair meeting exactly once), a valid switching of the data streams can always

be found-one in which each input data stream is switched onto a different output, and no two of the resulting streams pass through the same junction box. Additionally, give an algorithm to find such a valid switching.

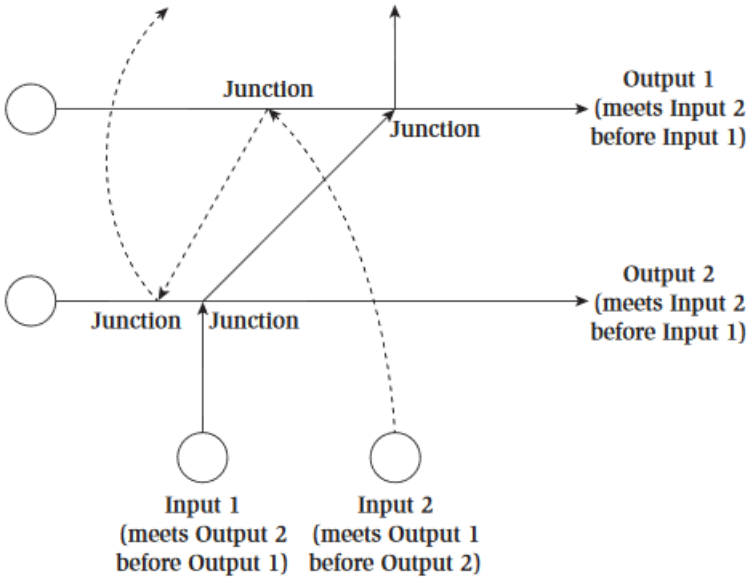

Figure 1.8 An example with two input wires and two output wires. Input 1 has its junction with Output 2 upstream from its junction with Output 1; Input 2 has its junction with Output 1 upstream from its junction with Output 2. A valid solution is to switch the data stream of Input 1 onto Output 2, and the data stream of Input 2 onto Output 1. On the other hand, if the stream of Input 1 were switched onto Output 1, and the stream of Input 2 were switched onto Output 2, then both streams would pass through the junction box at the meeting of Input 1 and Output 2—and this is not allowed.

4. Take the following list of functions and arrange them in ascending order of growth rate. That is, if function $g(n)$ immediately follows function $f(n)$ in your list, then it should be the case that $f(n)$ is $O(g(n))$.

$$g_1(n) = 2^{\sqrt{logn}}$$
$$g_2(n) = 2^n$$
$$g_3(n) = n^{4/3}$$
$$g_4(n) = n(logn)^3$$
$$g_5(n) = n^{logn}$$
$$g_6(n) = 2^{2^n}$$
$$g_7(n) = 2^{n^2}$$

5. Assume you have functions f and g such that $f(n)$ is $O(g(n))$. For each of the following statements, decide whether you think it is true or false and give a proof or counter example.

   (a) $log_2 f(n)$ is $O(log_2 g(n))$.

   (b) $2^{f(n)}$ is $O(2^{g(n)})$.

   (c) $f(n)^2$ is $O(g(n)^2)$.

6. Consider the following basic problem. You're given an array A consisting of n integers A[1], A[2],..., A[n]. You'd like to output a two-dimensional n-by-n array B in which B[i,j](for $i < j$) contains the sum of array entries A[i] through A [j]-that is, the sum A[i]+A[i+1]+...+A[j]. (The value of array entry B[i,j] is left unspecified whenever $i >= j$, so it doesn't matter what is output for these values.)

Here's a simple algorithm to solve this problem.

```
For i = 1, 2,..., n
    For j = i+1, i+2,..., n
        Add up array entries A[i] through A[j]
        Store the result in B[i,j]
    Endfor
Endfor
```

(a) For some function f that you should choose, give a bound of the form $O(f(n))$ on the running time of this algorithm on an input of size n (i.e., a bound on the number of operations performed by the algorithm).

(b) For this same function f, show that the running time of the algorithm on an input of size n is also $\Omega(f(n))$. (This shows an asymptotically tight bound of $\Theta(f(n))$ on the running time.)

(c) Although the algorithm you analyzed in parts (a) and (b) is the most natural way to solve the problem-after all, it just iterates through the relevant entries of the array B, filling in a value for each-it contains some highly unnecessary sources of inefficiency. Give a different algorithm to solve this problem, with an asymptotically better running time. In other words, you should design an algorithm with running time $O(g(n))$, where $lim_{n\to\infty} g(n)/f(n) = 0$.