SOFTENG 250

William Shin 884819645

Homework 1

1a) m*n

1b) Each company c can make a pair with each applicant a, i.e. n companies can pair with m applicants. So, there are m*n combinations of pairs.

2)
Exercise 1)

A perfect matching **M** has every member *c* of **C** and every member *a* of **A** appearing in exactly one pair **M**. There are n! perfect matchings because: a_1 has n options of *c* to pair with, a_2 has (n-1) options of *c* to pair with, … until a_n has 1 option of *c* to pair with. Therefore there are n*(n-1)*(n-2)*….*1 perfect matchings, i.e. n! perfect matchings.

Exercise 2)

C = $\{c_1, c_2, c_3\}$, **A** = $\{a_1, a_2, a_3\}$

Preference list for companies:

$c_1 = [a_1, a_2, a_3]$ $c_2 = [a_2, a_3, a_1]$ $c_3 = [a_3, a_2, a_1]$

Preference list for applicants

$a_1 = [c_1, c_2, c_3]$ $a_2 = [c_2, c_3, c_1]$ $a_3 = [c_3, c_2, c_1]$

Perfect matchings:

1) c_1a_1 c_2a_2 c_3a_3
2) c_1a_1 c_2a_3 c_3a_2
3) c_1a_2 c_2a_3 c_3a_1
4) c_1a_2 c_2a_1 c_3a_3
5) c_1a_3 c_2a_1 c_3a_2
6) c_1a_3 c_2a_2 c_3a_1

Stable matching:

c_1a_1 c_2a_2 c_3a_3

Exercise 3)

Let M_1 = (c', a) which is a matching. There are 2 two options. If *a* prefers c' to c, so after an iteration, $M_2 = M_1$, so M_2 is also a matching. Otherwise, if *a* prefers c to c', (c', a) is removed and (c, a) is added as part of the iteration. M_2 is still a matching because the definition of a matching is still met where each member of **C** and each member of **A** appears in at most one pair in M_2 since c' is removed from the set.

Exercise 4)

The algorithm iterates so that every company offers an internship to every applicant. The final iteration will form a pair between the last *c* and the remaining *a*. The algorithm works by creating a perfect matching M, i.e. no companies and applications are free.

Exercise 5)

Let the number of companies be *n* and the number of applicants be *m*. The input size would be $n + m + 2mn$.

3)

Exercise 1)

For every element e ($t_1, \ldots, t_n$),

If element equals 'exchange', 'escape', 'data', or 'stream',

Display word

endIf

endFor

Input size is n.

Exercise 2)

The algorithm produces a sorted array because it appends the smaller element out of the two elements pointed by the two pointers in A and B onto the output array until one of the lists become empty. When one of the lists become empty, the rest of the other array is appended onto the output array. Since the algorithm compares the two elements and only outputs the smaller value until the next iteration, the output array is sorted.

Exercise 3)

To correctly analyse the running time of an algorithm, we must look at the worst-case scenario. The worse case scenario for the Euclidean algorithm is when the inputs are two adjacent Fibonacci numbers. This is proven by Lamé's Theorem: "the Euclidean algorithm requires n steps when $a = f_{n+2}$ and $b = f_{n+1}$." Let T(a,b) be the number of steps, i.e. the time taken in the Euclidean algorithm, where $a \geq b$. When $a = f_{10} = 55$and $b = f_9 = 34$, T(55,34) = 8 steps. The iterations are:

55 = 34 * 1 + 21

34 = 21 * 1 + 13

21 = 13 * 1 + 8

13 = 8 * 1 + 5

8 = 5 * 1 + 3

5 = 3 * 1 + 2

3 = 2 * 1 + 1

2 = 1 * 2,

i.e. 8 steps.

4)

It is possible for all applicants to be unlucky if all the company's first choice of applicant has that company as their worst ranked company in their preference list.

For example, (3 applicants 3 companies):

Preference list for companies:

$c_1 = [a_1, a_3, a_2]$ $c_2 = [a_2, a_1, a_3]$ $c_3 = [a_3, a_2, a_1]$

Preference list for applicants:

$a_1 = [c_3, c_2, c_1]$ $a_2 = [c_1, c_3, c_2]$ $a_3 = [c_2, c_1, c_3]$

After three iterations, the stable matching is $[c_1a_1 \quad c_2a_2 \quad c_3a_3]$, and since there are no more free companies or applicants the algorithm ends. At this stage, the three applicants are all unlucky because the companies that they are matched with are respectively the worst ranked company in the applicant's preference list.

5)

True, the pair (c,x) belongs to M. If *c* ranks *x* highest and vice versa, *x* will always end up with *c*. If *c* makes *x* an internship offer, *x* will accept, and no other company can change the internship pair that *x* belongs to. If *c′* already has an internship pair with *x*, *x* will leave that pair and pair up with *c*.