Problem 1

Let C be a set consisting of n companies, and A be a set consisting of m applicants. Consider the set $C \times A$ of all ordered pairs of the form (c, a), where $c \in C$ and $a \in A$.

Part A and B

How many ordered pairs are there?, and explain your answer.

Solution

There would be $m \times n$ ordered pairs.

For each $c \in C$, there are m possible pairs in the form of (c, a) (a is an element of A, where the size of set A is m)

Since there are n elements in C, the size of $C \times A$ is $n \times m$

Problem 2

Solve all exercises in LN1.

Exercise 1. Explain why there are exactly $n!$ perfect matchings.

Since the first company gets n applicants to choose from and then the second company gets $n-1$ applicants to choose from, so there will be $n \cdot (n-1) \cdot (n-2) \cdot ... \cdot 1 = n!$ sets of perfect matchings.

Exercise 2. Assume that $C = \{c_1, c_2, c_3\}$ and $A = \{a_1, a_2, a_3\}$.

- Set up a preference list for all companies and all applicants.

Preference List:

$$c_1 : a_1 > a_2 > a_3 \qquad a_1 : c_1 > c_2 > c_3$$
$$c_2 : a_2 > a_1 > a_3 \qquad a_2 : c_2 > c_1 > c_3$$
$$c_3 : a_3 > a_1 > a_2 \qquad a_3 : c_3 > c_1 > c_2$$

- List all possible perfect matchings for your preference lists. List all stable matchings for your preference list.

Perfect Matching:

$M_1 = \{(c_1, a_1), (c_2, a_2), (c_3, a_3)\}$ **This is a stable matching**
$M_2 = \{(c_1, a_1), (c_2, a_3), (c_3, a_2)\}$ Not stable
$M_3 = \{(c_1, a_2), (c_2, a_3), (c_3, a_1)\}$ Not stable
$M_4 = \{(c_1, a_2), (c_2, a_1), (c_3, a_3)\}$ Not stable
$M_5 = \{(c_1, a_3), (c_2, a_1), (c_3, a_2)\}$ Not stable
$M_6 = \{(c_1, a_3), (c_2, a_2), (c_3, a_1)\}$ Not stable

Problem 2 continued on next page...

Exercise 3. In the last part of our argument above (in lecture notes) M_2 is obtained from M_1 by removing (c', a) from M_1 and adding (c, a) to M_1. Write down your reasons explaining why M_2 is a matching.

By definition: For M_k to be matching, it is required that every member $c \in C$ and every member $a \in A$ appears in at most one pair in M_k.

Since M_1 is matching and as it goes through the iteration we are removing a pair (c', a) and adding (c, a) to produce M_2, and since it is not necessary for c' to have a pair in M_2 for it to be matching we can conclude that M_2 is indeed matching.

Exercise 4. Finish the proof by showing that every applicant a must have a pair in M.

For this proof, we shall use proof by contradiction.

Assume that applicant a does not have a pair in M then after the last iteration we are left with one applicant who still doesn't have a pair however since it was the last iteration of the loop, it must have ended because all companies and applicants have matched (this is also assuming the fact that there are n companies and n applicants) therefore we have a contradiction because this would imply there are more companies than applicants which is not the case.

QED.

Exercise 5. We postulated that the number of companies equals the number of applicants. Suppose we remove this postulate. What will the size of input data be in this case?

Let $C = \{c_1, c_2, ..., c_m\}$ and $A = \{a_1, a_2, ..., a_n\}$

The size of the input data now will be $n + m + 2mn$ since we represent companies and applicants as single arrays of size n and m respectively and we represents the company and applicants preferences as a $m \times n$ double array.

Problem 3

Exercise 1. Write down an algorithm that given input $\{t_1, ..., t_n\}$ of texts, outputs those texts t that contain any of the following words: exchange, escape, data and stream. What do you think the size of the input is in this example?

Algorithm 1 findWords(T)

```
1: Initialize array of size n to store texts (Call this array r)
2: Initialize an array of size 4 to store words: exchange, escape, data and stream (Call this array w)
3: For i = 1 to n
4:     For j = 0 to 3
5:         - If any word w_j is found in t_i
6:             - Store t_i in r
7:     End inner for loop
8: End outer for loop
9: Return r
```

The input size would be the size of the input array T and how many texts it contains so in this case it would be n, however these texts may contain more than one word and hence encode more bytes - so if a text contains m words the input size could be $m \times n$

Problem 3 continued on next page...

Exercise 2. Explain why $Merge(A, B)$ - algorithm produces a sorted array. For simplicity you can assume that all elements in A and B are pairwise distinct.

If we look at the algorithm for Merge we can see that it takes two sorted arrays A and B and loops through the arrays until there are no elements left, since the algorithm compares the elements at the pointers - it always takes the smallest element out of the two lists and appends that one to the output list so our output list will always be sorted. Also in the case if both lists are different sizes, the largest sized list will automatically be appended to the end of the return list and since that list is sorted it will always output a sorted list.

Exercise 3. For this exercise, the sizes of n and m are the lengths of the decimal representations of n and m. Explain why Euclidean algorithm runs in time proportional to the size of the input. For the analysis assume that the division process takes a constant amount of time

Since we can assume the division process takes a constant time, and all other operations such as setting and if statements take constant time (i.e. $\mathcal{O}(c)$) and we can say that the while loop runs at an arbitrary time t so $\mathcal{O}(x)$ (where $x > 0$ and $x = f(n, m)$) so we can say that the whole algorithm will have a time complexity of $\mathcal{O}(c + x) = \mathcal{O}(x)$.

From this we can see that the number of times the while loop runs determines the complexity of the algorithm, furthermore since the loop depends on the inputs n , m we can say that $\mathcal{O}(x) = \mathcal{O}(n + m)$

Therefore the only thing that would change the time of the algorithm would be the input n and m and hence proportional.

Problem 4

Consider the GS-algorithm. Let M be the output of the algorithm. We know (from the lecture) that M is a stable matching.

Say that an applicant x is unlucky according to M if the matching M assigns company c to the applicant x so that c is the worst ranked company in the applicant's preference list. Is it possible that all applicant are unlucky according to M? If so, then give such an example with 3 applicants and 3 companies. If not, explain your answer in brief.

Consider the case where

Preference List:

$$c_1 : a_1 > a_2 > a_3 \qquad a_1 : c_3 > c_2 > c_1$$
$$c_2 : a_2 > a_3 > a_1 \qquad a_2 : c_1 > c_3 > c_2$$
$$c_3 : a_3 > a_1 > a_2 \qquad a_3 : c_2 > c_1 > c_3$$

From this we can see all companies have different applicants on the top of their preference list, and all these applicant don't rank these companies back very highly.

After the lists go through the algorithm it can be seen that this is the matching produced:

$$\{(c_1, a_1), (c_2, a_2), (c_3, a_3)\}$$

As the companies give out their offers and the applicants receive their offers they have to accept since there are no more offers because all applicants have to accept since it is the best option and thus all applicants are

stuck with their least preferred company and all companies have their most preferred applicant and thus all applicants are considered to be unlucky.

Problem 5

As above, let M be the output of the GS-algorithm. Assume that company c ranks an applicant x first; also assume that the applicant x, too, ranks c first. Does this imply that the pair (c, x) belongs to M? Answer the question as false or true, and explain your answer in brief.

True: Since company c has x at the top of it preference list, it means that the company doesn't prefer any other applicant at all and the same goes for the applicant, they only prefer c and c only therefore the algorithm will match them together and the pair (c, x) (since it is the best offer) will indeed be in M and hence it will be a stable matching.

Using an integral approximation of the sum of the logarithms of the first n natural numbers gives:

$$n\log n - n = \int_1^n \log(t)dt \leq \underbrace{\sum_{k=1}^{n}\log(n)}_{\log(n!)} \leq \int_1^{n+1}\log(t)dt = \underbrace{(n+1)\log(n+1)-(n+1)}_{n\log(n)-n-1+\log(n)+(n+1)\log(1+1/n)} \implies n\log(n)-n \leq \log(n!) \leq n\log$$

However we have that:

$$-\frac{3\log(n)}{n} \leq -\frac{\log(n)}{n} - \underbrace{2\log(e^{1/n})}_{\frac{2}{n}} \leq -\frac{\log(n)}{n} - 2\log(1+1/n) \leq \underbrace{\frac{1}{n} - \frac{\log(n)}{n} - \frac{n+1}{n}\log(1+1/n)}_{-\frac{\delta(n)}{n}} \implies -\frac{3\log(n)}{n} \leq -\frac{\delta(n)}{n} \implies$$

Which by our first inequality gives us:

$$1 - \frac{3\log(n)}{n} \leq \frac{\log(\frac{n^n}{n!})}{n} \leq 1 \implies 1 - 3\underbrace{\left(\lim_{n\to\infty}\frac{\log(n)}{n}\right)}_{0} \leq \lim_{n\to\infty}\frac{\log(\frac{n^n}{n!})}{n} \leq 1 \implies 1 \leq \lim_{n\to\infty}\frac{\log(\frac{n^n}{n!})}{n} \leq 1 \implies \lim_{n\to\infty}\frac{\log(\frac{n^n}{n!})}{n} = 1$$

As required.