Written Exam

10:00 – 12:30, February 6, 2018

Entrance Examination (AY 2018)

Department of Computer Science
Graduate School of Information Science and Technology
The University of Tokyo

Notice:

(1) Do not open this problem booklet until the start of the examination is announced.

(2) Answer the following 4 problems. Use the designated answer sheet for each problem.

(3) Do not take the problem booklet or any answer sheet out of the examination room.

Write your examinee's number in the box below.

Examinee's number	No.

Problem 1

In this problem, $\mathbb{R}$ represents the set of real numbers, and $\mathbb{R}^N$ represents the set of real column vectors of length N. For $\boldsymbol{v} \in \mathbb{R}^N$, $\boldsymbol{v}^\top$ denotes its transpose. Let I be the $N \times N$ identity matrix.

Consider an eigensystem of a real $N \times N$ symmetric matrix A,

$$A\boldsymbol{x} = \lambda \boldsymbol{x},$$

where λ and $\boldsymbol{x}$ are an eigenvalue and a corresponding eigenvector, respectively.

Let $\lambda_{\max}(M)$ be the maximum of eigenvalues of matrix M.

You may use the following facts on the eigenvalues and the eigenvectors of a real $N \times N$ symmetric matrix without proofs;

- There are N independent eigenvectors that form an orthogonal basis.
- Every eigenvalue is a real number.

Answer the following questions.

(1) Prove that if $\boldsymbol{x}$ is an eigenvector of A, it is also an eigenvector of $A + \mu I$ for any $\mu \in \mathbb{R}$.

(2) Prove that

$$\lambda_{\max}(A) = \max\{\boldsymbol{v}^\top A \boldsymbol{v} \mid \boldsymbol{v} \in \mathbb{R}^N,\ \boldsymbol{v}^\top \boldsymbol{v} = 1\}.$$

(3) Prove that

$$\boldsymbol{v}^\top \left(\lambda_{\max}(A) I - A\right) \boldsymbol{v} \geq 0$$

for any $\boldsymbol{v} \in \mathbb{R}^N$.

(4) Suppose that matrix B is also an $N \times N$ real symmetric matrix. Prove that

$$\lambda_{\max}(A + B) \leq \lambda_{\max}(A) + \lambda_{\max}(B).$$

Problem 2

For each $n \geq 1$, let Σ_n be $\{a_1, \ldots, a_n\}$, where a_1, ..., a_n are different from each other. For a word $w \in \Sigma_n^*$, we write $|w|_{a_i}$ for the number of occurrences of a_i in w. We define the languages $L_{\forall,n}$ and $L_{\exists,n}$ over Σ_n by:

$$L_{\forall,n} = \{w \in \Sigma_n^* \mid |w|_{a_i} \text{ is even for every } i \in \{1, \ldots, n\}\},$$

and

$$L_{\exists,n} = \{w \in \Sigma_n^* \mid |w|_{a_i} \text{ is even for some } i \in \{1, \ldots, n\}\}.$$

Answer the following questions.

(1) Give a *deterministic* finite state automaton with 4 states that accepts $L_{\forall,2}$.

(2) Give a *non-deterministic* finite state automaton with 7 states (without ϵ-transitions) that accepts $L_{\exists,3}$.

(3) Prove that, for every $n \geq 1$, every *deterministic* finite state automaton that accepts $L_{\exists,n}$ has at least 2^n states.

(4) Prove that, for every $n \geq 1$, every *non-deterministic* finite state automaton (without ϵ-transitions) that accepts $L_{\forall,n}$ has at least 2^n states.

Problem 3

Suppose that we have a set of 2^N elements and its partition into subsets where every element belongs to one and only one of the subsets. We want to support the following two operations for a partition.

`FIND(x)`	identifies the subset that element `x` belongs to.
`MERGE(A,B)`	merges two subsets, `A` and `B`.

We use a forest-of-trees structure, where each subset forms a tree. Each tree node corresponds to an element and has a pointer to its parent. The pointer of a root node points to the identity of the subset it belongs to. `FIND(x)` operation traces pointers from node `x` to the root. `MERGE(A,B)` operation changes the pointer of the root node of subset `A` so that it points to the root of subset `B`.

We initially have 2^N subsets, where each subset contains a single element. We then repeatedly merge a pair of subsets until we get a single subset containing all the elements. Height of a tree is defined as the number of edges on the longest path between its root and a leaf.

Answer the following questions.

(1) How many merge operations are required to merge all the subsets?

(2) What is the minimum (best case) tree height after the completion of all the merge operations among all the possible merge sequences? Also explain why.

(3) What is the maximum (worst case) tree height after the completion of all the merge operations among all the possible merge sequences? Also explain why.

(4) One can reduce the maximum (worst case) tree height by slightly modifying the `MERGE(A, B)` operation. Explain how to modify the operation. Also, give the maximum tree height when using the modified operation, with a brief explanation.

(5) One can reduce the height of a tree without increasing computational complexity, by performing an additional procedure when applying the `FIND(x)` operation to an element `x` in the tree. Explain how.

Problem 4

In this problem, we consider mutual exclusion of concurrent processes running on a multiprocessor system. Assume that the execution of the code `x = x + 1` consists of the following three operations: (i) load the initial value of `x` to a register R from a memory address A, (ii) add 1 to R, and (iii) store the value of R to A.

Answer the following questions.

(1) Consider the case where two processes share a variable `x` and execute `x = x + 1` concurrently on this multiprocessor system without mutual exclusion. Assuming that the initial value of `x` is 0, answer all the possible values of x after both the processes complete the executions of `x = x + 1`.

(2) A standard way to achieve mutual exclusion of the executions of `x = x + 1` is to use the `TestAndSet` instruction as in the following C code.

```
while (TestAndSet(&lock));
x = x + 1;
lock = 0;
```

Here, `x` and `lock` are shared variables, whose initial values are 0. The `TestAndSet` instruction, with a hardware support, atomically executes the functionality that is described by the following C code. Answer appropriate expressions that fill the blanks from (A) to (E).

```
int TestAndSet(int *a) {
    int b;
    (A) = (B);
    (C) = (D);
    return (E);
}
```

(3) An alternative way to achieve mutual exclusion is to use another atomic instruction `Swap`, whose functionality is described by the following C code.

```
void Swap(int *a, int *b) {
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
```

Using the `Swap` instruction, mutual exclusion of the executions of `x = x + 1` can be achieved as follows.

```
int key = (F);
while ((G) == 1)
    Swap((H), (I));
x = x + 1;
lock = 0;
```

Here, `x` and `lock` are shared variables whose initial values are 0, and `key` is a local variable. Answer appropriate expressions that fill the blanks from (F) to (I).