Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

FS 2025

Prof. R. Wattenhofer

Principles of Distributed Computing
Sample Solution to Exercise 10

1 Determining the Median

As stated in the hint, we start with initializing the nodes to give them IDs from $1, \dots, n$. Then, the node with ID i transmits its token in time slot i. Each node uses two integer variables n^-, n^+ to count how many other nodes have a token that is smaller/bigger than its own token. Once a node has received the tokens of all other nodes, it just needs to check if $n^+ = n^-$. The node whose token is the median can simply broadcast it afterwards.

Let us briefly analyze the time and space used by this. We know from the lecture that initialization takes $\mathcal{O}(n)$ rounds. Sending all tokens afterwards takes another n rounds, i.e., does not increase the asymptotic runtime.

Each node needs to store a new ID of up to n, which takes up $\mathcal{O}(\log n)$ space. The variables n^-, n^+ from the algorithm can also be stored by using $\mathcal{O}(\log n)$. Note that we do not need to store the values of the tokens of other nodes. Therefore, we only need $\mathcal{O}(\log n)$ space in total.

The correctness follows by the construction of the algorithm. The unique node whose upper and lower counter has the same value can broadcast its token and thus all nodes are aware that it is the median.

Remark: We can further reduce the memory usage if, instead of storing n^+ and n^-, we only store the difference $n^+ - n^-$ in a variable called η. Every time a node receives a token which is bigger than its own token, it increments η. If a node receives a token smaller than its own token, it decrements η. The median of all tokens will then be found in the node which has $\eta = 0$.

Remark: Note that the lecture script only claims that our uniform initialization algorithm finishes in expected time $\mathcal{O}(n)$. However, one can also show that this algorithm finishes in $\mathcal{O}(n)$ time w.h.p., and with a slight modification, we can ensure that the nodes use no more than $\mathcal{O}(\log n)$ space in this algorithm. Hence we can indeed begin our solution with the initialization of the nodes.

2 Maximum

See Algorithm 1 below for a pseudocode description of the algorithm. We will now proceed to analyze the runtime of this algorithm.

A round is *good* if at least half the vertices exit in that round. Observe that at most $log(n)$ good rounds are needed to find the maximum. Since the leaders are chosen randomly and independently in each round, a round is good with probability at least $\frac{1}{2}$. Let the random variable G be the number of good rounds after $4c \cdot \log(n)$ rounds, for $c \geq 1$. The expectation of G is bounded by

$$E[G] \geq \frac{1}{2} 4c \cdot \log(n) = 2c \cdot \log(n) \geq 2 \cdot \log(n)$$

Algorithm 1 Algorithm for determining the maximum

```
while TRUE do
    elect leader
    leader broadcasts value
    if own value is bigger then
        broadcast own value
    end if
    if no transmitter then
        leader has max
    else if single transmitter then
        transmitter has max
    else if own value $\leq$ leader then
        exit
    end if
end while
```

Using the Chernoff bound and setting $\delta = \frac{1}{2}$ we get the following as probability of failure (i.e., not having enough good rounds):

$$\begin{aligned} Pr[G < \log(n)] &= Pr[G < (1-\delta)2 \cdot \log(n)] \\ &\leq Pr[G < (1-\delta)E[G]] \\ &\leq e^{-\frac{\delta^2}{2}E[G]} \\ &\leq e^{-\frac{1}{4}c\log(n)} \\ &= n^{-\frac{c}{4}} \end{aligned}$$

Additionally, we have a failure probability of at most $n^{-c'}$ every time we elect a leader. Note that we look at the probability of succeeding in $4c \cdot \log(n)$ rounds, and we perform one leader election in each of these rounds. We set $c' = \frac{c}{4}$; this is possible, since our leader election algorithm allows us to reduce the probability of failure to $n^{-c'}$ for any constant c' of our choice.
With the union bound we can derive

$$\begin{aligned} Pr[\text{fail}] &= P\left[\{G < \log(n)\} \cup \bigcup_i \{\text{leader election in round } i \text{ failed}\}\right] && \mid \text{Union Bound} \\ &\leq n^{-c'} + 4c \cdot \log(n) n^{-c'} \\ &= (1 + 4c \cdot \log(n)) n^{-c'} && \mid n > 1, c \geq 1 \\ &\leq 5c \cdot \log(n) n^{-c'} \\ &= 2^{\log(5c)} \cdot \log(n) n^{-c'} \\ &< 2^{\log(5c) \cdot \log(n)} n^{-c'} \\ &= n^{-c' + \log(5c)} \\ &= n^{-\alpha} \end{aligned}$$

where α is independent of n and can be set arbitrarily high by choosing the constant c'.