Problem Set 6 Solutions

This problem set is due **at 11:59pm** on **Thursday, October 28, 2016.**

EXERCISES (NOT TO BE TURNED IN)

Sortng Networks:

- CLRS 28.1-1, page 638
- CLRS 28.1-3, page 638

Bitonic Network:

- CLRS 28.3-2, page 647
- CLRS 28.4-3, page 648

Problem 6-1. Chopsticks [50 points]

You're eating dinner with three of your friends. For some reason, you only have four chopsticks for the four of you to eat with. Fortunately, you're all fine with sharing. However, when you try to take the chopsticks to eat, you all end up with one chopstick each. Now none of you can eat.

You realise that you need to design a mutual exclusion procedure to solve this problem.

The operations the people can perform are to take an unused chopstick, to eat, to give up a chopstick, and to acquire and release semaphores. If there aren't any chopsticks available, trying to take one will accomplish nothing.

A semaphore is a basic object with two operations: Acquire, which waits for a specific semaphore or one of a group of semaphores to become available, then takes control of it and makes it unavailable to others, and Release, which makes it available to others once again.

You'd like the protocol to guarantee the following:

Safety: No one eats with less than 2 chopsticks.

Liveness: Infinitely often, someone must eat. In other words, it should be impossible for every person to get stuck and never eat again.

Fairness: It's possible for each person to eat, eventually.

Efficiency: It's possible for more than one person to eat at once.

(a) [25 points] Your friend Ben Bitdiddle comes up with the following procedure:

There will be four semaphores: s_1, s_2, s_3, s_4.

Everyone will do the following, in a loop:

1. Acquire any available semaphore. Call it s_a.
2. Take a chopstick.
3. Acquire another available semaphore. Call it s_b.
4. Take another chopstick.
5. Eat.
6. Give up both chopsticks.
7. Release semaphore s_a.
8. Release semaphore s_b.

Which of the 4 guarantees does this procedure satisfy? Give an example or brief proof for each.

Solution: This satisfies Safety, Fairness and Efficiency.

Safety proof: The procedure ensures that no person ever has or attempts to have more chopsticks than semaphores, so the fact that there are only 4 semaphores means that no one ever tries to pick up a chopstick when none is available. Since the number of chopsticks taken and given up in a cycle is equal, the person always starts the loop

with 0 chopsticks. This means that when the person eats, at line 5, they have exactly 2 chopsticks.

Fairness and Efficiency demonstration: Suppose persons 1 and 2 go through a complete loop, then persons 3 and 4 go through a complete loop, and this alternates. Then everyone's eating, and 2 people are eating at once.

Liveness counterexample: Suppose everyone goes to step 2. Then every semaphore is taken, so no one can proceed to step 3.

(b) [25 points] Design your own procedure that satisfies all four properties. Briefly prove that it satisfies each guarantee.

Solution: Example solution:

Make two semaphores, s_1 and s_2. Persons 1 and 2 have s_1 assigned to them. Persons 3 and 4 have s_2 assigned to them.

Each person does the following, in a loop:

1. Acquire your assigned semaphore.
2. Take 2 chopsticks.
3. Eat.
4. Give up 2 chopsticks.
5. Release assigned semaphore.

Safety holds because each semaphore is worth 2 chopsticks, the way each semaphore was worth 1 chopstick in part a. In particular, twice number of semaphores that no one controls plus the number of chopsticks help is always 4 or less, which means that the number of chopsticks help is always 4 or less.

Fairness and Efficiency hold because 1 & 3 and 2 & 4 can alternate.

Liveness holds because either someone has a semaphore, in which can they can make progress, or no one has a semaphore, in which case someone can acquire a semaphore.

Problem 6-2. Safety and Liveness and Amdahl's Law [50 points]

(a) [24 points]

For each of the following, state whether it is a safety or liveness property. Identify the bad or good thing of interest (the guarantee that the safety or liveness condition is making)

1. Patrons are served in the order they arrive.
2. What goes up must come down.
3. If two or more processes are waiting to enter their critical sections, at least one succeeds.

4. If an interrupt occurs, then a message is printed within one second.
5. If an interrupt occurs, then a message is printed.
6. The cost of living never decreases.
7. Two things are certain: death and taxes.
8. You can always tell a Harvard man.

Solution:

1. *Safety:* serving patrons out of order is a bad thing which must never happen.
2. *Liveness:* eventually the coming-down state transition occurs.
3. *Liveness:* eventually one thread that wants in gets in.
4. *Safety:* we will not enter the bad state where no message has occurred but an interrupt happened more than a second ago.
5. *Liveness:* eventually the message will be printed.
6. *Safety:* You never enter the (bad?) state where you pay less to stay alive.
7. *Liveness:* ironically enough. Everyone eventually undergoes these two transitions.
8. *Safety:* You never enter a state where you are in the proximity of an unrecognized Harvard man.
   This can also be interpreted as a liveness guarantee:
   You eventually recognize every Harvard man.

(b) [26 points] You are given a program of which 25% is inherently sequential and the remaining 75% is fully parallelizable. You must purchase at least one processor to execute the program. However, a salesman for SQRT-Discount Corporation offers you up to $n = 7$ CPUs for the cost of $\$100\sqrt{n}$.

How many CPUs should you purchase to optimize the objective function $cost \times running\ time$? Explain in sufficient detail to establish that your answer is correct.

Solution: Amdahl's law says that the speedup **S** for application with **p** parallel fraction, and **n** processors is:

$$S = \frac{1}{1 - p + \frac{p}{n}} \tag{1}$$

The execution time **T** is $\frac{1}{S}$, and the cost **C** is $100\sqrt{n}$. We want to optimize the following function:

$$FUNC = C * T = 100\sqrt{n} * (\frac{1}{S}) \tag{2}$$

By plugging the $p = \frac{3}{4}$ we get:

$$FUNC = C * T = 100\sqrt{n} * (\frac{1}{S}) = 25 * (\sqrt{n} + \frac{3}{\sqrt{n}}) \tag{3}$$

The function is optimal for $n = 3$ (Since we have the product of positive terms being constant, the sum minimizes when terms are equal $\sqrt{n} = \frac{3}{\sqrt{n}}$).
Therefore, we should buy 3 processors from the SQRT-Discount company.

Problem 6-3. Weak balancer from matching [50 points]

A k-smooth sequence is one where the difference between any two elements is no more than k. In other words, if the sequence is $y_0, y_1, \ldots, y_{n-1}$, the sequence is k-smooth if $\forall i, j$ s.t. $0 \leq i, j < n, |y_i - y_j| \leq k$.

A matching layer of balancers for two sequences of equal length is one where the elements of the sequences are paired in a one-to-one correspondence, and each pair is passed as the inputs to a balancer.

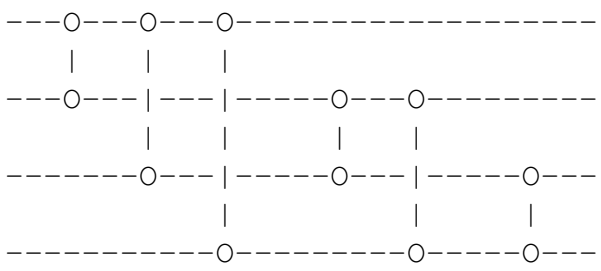
(a) [25 points] Let X and Y be two k-smooth sequences of length n. Suppose X and Y are passed as input to a matching layer of balancers. Let Z be the output of that matching layer. Show that Z is $(k + 1)$-smooth.

Solution: Let x, y be the minimum values of X, Y. Then every element in X, Y is at most $x + k, y + k$ respectively.
Every value in Z is at least $\left\lfloor \frac{x+y}{2} \right\rfloor$, and at most $\left\lceil \frac{(x+k)+(y+k)}{2} \right\rceil = k + \left\lceil \frac{x+y}{2} \right\rceil \leq k + 1 + \left\lfloor \frac{x+y}{2} \right\rfloor$. So Z is $(k + 1)$-smooth.

(b) [25 points] The bitonic balancing network guarantees a 1-smooth output on an input of length n, with depth $O((\log n)^2)$, and thus delay $O((\log n)^2)$. Sometimes, we might be willing to sacrifice some smoothness to make the algorithm run faster. Use (a) to construct a balancer network whose output is guaranteed to be $(\log_2 n)$-smooth, and which has $o((\log n)^2)$ depth (That is, the depth should grow with respect to n strictly slower than $(\log n)^2$).
Note: You may assume that $n = 2^m$, for some integer m.

Solution: We will use a recursive construction. A merge network of size n consists of two merge networks of size $n/2$, fed into a matching layer of size n.
We will prove the output is $\log_2 n$ smooth by induction. Assume that a network at depth d from the input has an output which is d-smooth. A pair of such networks are combined at depth $d+1$, giving an output which is $(d+1)$-smooth by part a. The base case is that the inputs at depth zero, namely the input values considered as separate sequences, are each 0-smooth. Thus we can conclude that networks at depth d are d-smooth. The overall network has depth $\log_2 n$, since the size of the sequences double with every merge and its output is size n. So the network is $\log_2 n$ smooth.
The final output is $(\log_2 n)$-smooth, and likewise the circuit is depth $\log_2 n$, which satisfies the problem.

Problem 6-4. Triangle Network [50 points] The drawing below is a Triangle network with width 4. In general, a Triangle(n) network has width n and $n(n-1)/2$ balancers/comparators, shaped into $n-1$ triangles of descending size. The ith triangle contains $n-i$ comparators/balancers connecting wires i, j, for $i = 1, 2, \ldots, n-1$ and for $j = i+1, i+1, \ldots, n$.



(a) [25 points] Prove that the Triangle(n) is **not** a counting network for some specific n.

Solution: See the size 4 example above. Suppose 4 tokens are input on the bottom wire. Then 2 end up on the top wire, while at least one of the other three wires ends up empty. This is a difference of 2, so the network is not a counting network.

(b) [25 points] Prove that the Triangle(n) network is a sorting network for all n.

Solution: This network is bubble sort.

It sorts by induction on the number of wires. On 1 wire, it trivially sorts. Assume it sorts on n wires. On $n+1$ wires, in the first triangle, it starts by comparing everything with the first wire, which must move the maximum to the first wire. The rest of the setup is Triangle (n) on the other n wires, so it sorts them. So the overall list is sorted.