Do your own work. Be neat and concise.

1. The figure below (similar to the figures on pages 3-23 and 3-24 of the lecture notes) shows the state of a simple crossbar scheduler at the start of a scheduling operation. Show the state of the controller after each step of the algorithm, until all possible matches have been made. Be sure to show how the pointers are updated. Do this for both the round robin algorithm and the i-SLIP algorithm.

![Crossbar Scheduler Diagram](image-url)

The result for the round robin algorithm is shown below.

![Round Robin Result Diagram](image-url)
The result for the i-SLIP algorithm is shown below.
2. The figure below (similar to the figure on page 3-31 of the lecture notes) shows the initial state of a CCF crossbar scheduler at the start of a scheduling operation. Show the state of the controller after each step of the algorithm.

<table>
<thead>
<tr>
<th>new arrivals</th>
<th>initial state</th>
</tr>
</thead>
<tbody>
<tr>
<td>b8</td>
<td>c4 b3</td>
</tr>
<tr>
<td>h8</td>
<td>a3 e6</td>
</tr>
<tr>
<td>c8</td>
<td>b7 h4 d3</td>
</tr>
<tr>
<td>a8</td>
<td>e7</td>
</tr>
<tr>
<td>d8</td>
<td>d6 a5 h5</td>
</tr>
<tr>
<td>b8</td>
<td>a3 b2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>following insertion of arriving cells</th>
<th>after 1st step</th>
</tr>
</thead>
<tbody>
<tr>
<td>b8 c4 b3</td>
<td>a 4</td>
</tr>
<tr>
<td>a3 h8 e6</td>
<td>b 2</td>
</tr>
<tr>
<td>b7 h4 c8 d3</td>
<td>c 1</td>
</tr>
<tr>
<td>a8 e7</td>
<td>d 1</td>
</tr>
<tr>
<td>d6 a5 d8 h5</td>
<td>e 2</td>
</tr>
<tr>
<td>b8 a3 b2</td>
<td>h 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>after 2nd step</th>
<th>after 3rd step</th>
</tr>
</thead>
<tbody>
<tr>
<td>b8 c4 b3</td>
<td>a 4</td>
</tr>
<tr>
<td>a3 h8 e6</td>
<td>b 2</td>
</tr>
<tr>
<td>b7 h4 c8 d3</td>
<td>c 1</td>
</tr>
<tr>
<td>a8 e7</td>
<td>d 1</td>
</tr>
<tr>
<td>d6 a5 d8 h5</td>
<td>e 2</td>
</tr>
<tr>
<td>b8 a3 b2</td>
<td>h 1</td>
</tr>
</tbody>
</table>

The steps are illustrated above. Following the third step, the algorithm terminates because a stable matching has been found.
3. (20 points) In one variant of the LOOFA crossbar scheduling algorithm, outputs that receive multiple “bids” from inputs, select inputs based on the timestamps of the contending cells (cells with small timestamps are preferred over cells with larger timestamps). Also, cells are forwarded from the output-side queues in timestamp order. Find a traffic pattern that demonstrates that even with a speedup of 2, this version of the LOOFA scheduling algorithm does not always forward cells in FIFO order.

The figure below illustrates a scenario in which the “oldest cells first” version of the LOOFA algorithm fails to preserve FIFO ordering. This is for a six port switch.
4. Show that the stable matching algorithm described on page 3-30 does in fact produce a stable matching.

Assume, to the contrary, that the constructed matching is not stable. This means that there are two pairs \((a_1, b_1)\) and \((a_2, b_2)\) such that \(a_1\) prefers \(b_2\) to \(b_1\) and \(b_2\) prefers \(a_1\) to \(a_2\). If this were the case, then \(a_1\) must have made a bid for \(b_2\) before it bid for \(b_1\). Since the members of \(B\) only change partners to improve their preference, at the time that \(a_1\) bid for \(b_2\), either \(b_2\) was unmatched or it was matched with a partner that it ranked no higher than \(a_2\). In this case, \(b_2\) must have switched partners, matching it with \(a_1\). But this yields a contradiction, since \(b_2\) would not have switched from \(a_1\) to \(a_2\), after becoming matched with \(a_1\).

5. (50 points) On the web site, you will find a program \texttt{xbStressPIM} that was used to produce the results on pages 3-21 and 3-22 of the lecture notes. Create a new version of this program that implements the i-SLIP scheduling algorithm, instead of the random matching algorithm used in the original. Your program should allow the number of iterations to be limited to a specified number. Use your program to produce charts like those on 3-21 and 3-22 for i-SLIP. Provide four sets of charts: the first set should limit the algorithm to one iteration, the second set should limit it to two iterations, the third to four and the last set should not limit the number of iterations. Compare your charts to the ones in the notes. Explain any differences that you observe.

The changes to the program are in the \texttt{doXfer()} subroutine, which is shown below.

```c
void doXfer(int b) {
    // Simulate transfer of cells from VOQs to outputs.
    int h, i, j, ii, jj, round;
    int req[MaxN][MaxN];    // req[i][j] indicates request from i to j
    int iMatch[MaxN];      // iMatch[i]=1 when input i has been matched
    int oMatch[MaxN];      // oMatch[j]=1 when output j has been matched
    static int iNxt[MaxN];  // iNxt[i] is next output to be considered at
    static int oNxt[MaxN];  // oNxt[j] is next input to be considered at
    // input i
    static int flag = 0;

    if (flag == 0) {
        for (i = 0; i < n; i++) {
            iNxt[i] = randint(0,n-1);
            oNxt[i] = randint(0,n-1);
        }
        flag = 1;
    }

    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            req[i][j] = (voq[i][j] > 0 ? 1 : 0);
            iMatch[i] = oMatch[i] = 0;
        }
    }

    for (round = 1; round <= b; round++) {
        for (j = 0; j < n; j++) { // unmatched outputs select inputs
            if (oMatch[j]) continue;
            for (i = 0; i < n; i++) {
                ii = (i + oNxt[j]) % n;
                if (req[ii][j] == 1 && iMatch[ii] == 0) {
                    req[ii][j] = 2;
                    break;
                }
            }
        }
    }
}
```
for (i = 0; i < n; i++) { // unmatched inputs select outputs
    if (iMatch[i]) continue;
    for (j = 0; j < n; j++) {
        jj = (j + iNxt[i]) % n;
        if (req[i][jj] == 2 && oMatch[jj] == 0) {
            iMatch[i] = 1; oMatch[jj] = 1;
            voq[i][jj] -= 1.0;
            voqSum[jj] -= 1.0;
            oq[jj] += 1.0;
            if (round == 1) {
                iNxt[i] = (jj + 1) % n;
                oNxt[jj] = (i + 1) % n;
            }
            break;
        }
    }
}
}
The first pair of charts appears below. This is for i-SLIP with one iteration (1-SLIP). The results are similar to, but not as good as, the results for Parallel Iterative Matching that are given in the lecture notes. One plausible explanation for this is the limitation of i-SLIP to one iteration. However, we will see later that even with an unlimited number of iterations, i-SLIP doesn’t perform quite as well in this case. If one considers the differences between the behavior of the two algorithms on the stress test, the reason becomes apparent. With i-SLIP all VOQs at input 0 get exactly one fifth of the switch bandwidth during the period around time 60,000. However, PIM behaves a little differently. Since inputs 0-2 are all competing for outputs 0-3, input 0 sees only 1/3 of the “requests” from these outputs, while it sees all the requests from output 4. Consequently, VOQ(0,4) gets a larger share of the bandwidth at input 0 than do the other VOQs. This means that it does not accumulate as large a backlog during the period around 60,000 as occurs when the i-SLIP algorithm is used.
The second set of charts are shown below. This set allows two iterations of the i-SLIP algorithm. Surprisingly, there is no improvement in performance for the stress test traffic. Specifically, the size of the backlog produced in the last phase is identical to the backlog produced for a single iteration. It appears that for the given traffic, a single iteration is sufficient to transfer as much traffic through the crossbar as can be transferred. However, there is an interesting difference in the output queue curves for a speedup of 1.5 (second chart from the top). The curve for output 3, which receives traffic in the penultimate phase, has a peculiar shape. It first rises, then drops to zero before spiking up again. It is difficult to explain this, but it appears that in the period just before time 60,000 inputs 1 and 2 transfer their packets to output 1 at a higher rate than they send to the other outputs, causing their VOQs for output 1 to empty first. This allows inputs 1 and 2 to send then to output 3 at a higher rate, causing the final spike in the traffic to output 3. In this part of the stress test, the i-SLIP algorithm seems to divide the bandwidth at the inputs unevenly among the different VOQs and this effect allows the packets going to output 1 to be transferred at a higher rate than they would be if the bandwidth were divided evenly.
The chart below shows the results for four iterations. It is identical to the previous results. The results for 16 iterations are also identical and are omitted.
6. One way to implement multicast in a crossbar switch with VOQs is for the IPPs to copy each arriving multicast cell to the VOQs for all outputs that are to receive copies. With this approach, the multicast cells appear no different than unicast cells to the crossbar. Show that a crossbar can forward multicast cells with a fanout of $F$ in a work-conserving fashion, using the LOOFA algorithm and a speedup of $F+1$.

We start by assuming that the switch operates in cycles of $F+3$ phases. The first phase of each cycle is the arrival phase, during which an arriving cell may be placed in up to $F$ different VOQs. The next $F$ phases are transfer phases, during which cells are transferred from inputs to outputs through the crossbar. The next phase is the departure phase, during which cells are transmitted on the output links. The last phase is an additional transfer phase.

We define the slackness of a cell just as on page 3-26 of the notes. We note that the lemma on page 3-26 generalizes the multicast case. As with the unicast case, each transfer cycle, during which a cell remains on the input side of the crossbar, increases the cell’s slackness by at least one. During the arrival phase, a cell’s slackness can decrease by at most $F$ and during the departure phase its slackness can decrease by at most 1. Since there are $F+1$ transfer phases, there can be no net decrease in a cell’s slackness during an $F+3$ phase cycle.

We can also give a version of lemma on 3-27. In particular, we can show that following the arrival phase of any step, the slackness of each cell is at least $-(F-1)$. For cells that arrive during the first time step, this is clearly true, since the outputs have no cells at this point, and at most $F$ cells get inserted into VOQs during the first step. So, the cell which is last in the ordering has a slackness of $-(F-1)$. As on 3-27, we proceed by induction. Let $c$ be a cell that was placed in a VOQ during the arrival step of phase $t$. If there are no cells that precede $c$ and were present at the input before step $t$, then the result clearly holds, since only the cells that arrived during step $t$ could precede $c$ and there are at most $F-1$ of these, excluding $c$. So, suppose $b$ is a cell that was present before step $t$ and that $b$ precedes $c$. Assume also, that all other cells present before step $t$ that precede $c$ also precede $b$. By the induction hypothesis, following the arrival phase of step $t-1$, the slackness of $b$ was at least $-(F-1)$. So, just before the arrival phase of step $t$, the slackness of $b$ must be at least 1. Now, consider the cells placed in VOQs during step $t$. Let $i$ be the number of these cells that precede $b$. This means that after the arrival phase of step $t$, the slackness of $b$ is at least $1-i$. Since there are at most $(F-1)-i$ other cells that arrived during step $t$ that precede $c$ and do not precede $b$, the slackness of $c$ is at least $(1-i)-(F-1)-1=-(F-1)$.

To complete the proof that the system is work-conserving, suppose that after the arrival phase of some step, there is some output with no cells in its output queue and there are cells waiting in VOQs for that output. Let $c$ be any such cell. Since there are no cells in the output queue, and since the slackness of $c$ is at least $-(F-1)$, there can be at most $F-1$ cells that precede $c$ at its input. This means that during each of the $F$ transfer phases between the arrival phase and the departure phase, either some cell in front of $c$ gets transferred, or the input attempts to send $c$. Since there only $F-1$ cells in front of $c$, there must be at least one transfer phase during which the input attempts to send $c$. If $c$ does not get sent during such a transfer phase, then the output must have gotten a cell from some other input. So, the output has a cell to send, during the departure phase.

7. Consider a multistage network in which each switch element has four inputs and four outputs, and there is an eight slot output queue associated with each output. Suppose the four queues contain 2, 5, 6 and 7 cells, respectively. If the system uses grant flow control,
how many grants can be sent to the upstream neighbors, at the start of the cell cycle? Why?

Only 1 grant can be sent, because the receiving SE must allow for the worst-case situation in which all the incoming cells are addressed to output 3, which only has room for 1 more cell.

Suppose the system uses acknowledgement flow control and each of the upstream neighbors sends a cell to the given switch element. What is the maximum number of cells that may be acknowledged by the switch element in this situation? Why? What is the minimum number? Why?

Up to 4 cells could be acknowledged. This will occur if no more than one of the incoming cells needs to use output 3, no more than 2 need to use output 2 and no more than 3 need to use output 1. Any set of incoming cells that satisfies all these conditions can be stored in the output queues, so all will be acknowledged. The minimum number is 1. This occurs if all arriving cells need to use output 3. In this case only one of them can be acknowledged.

Suppose that the switch element uses a single shared buffer with a total capacity of 24 cells, and the queue contains 2 cells for output 0, 5 cells for output 1, 6 for output 2 and 7 for output 3. In this case, how many grants will be given to the upstream neighbors, if grant flow control is used? Why? If acknowledgement flow control is used, how many acks will be given? Why?

4 grants may be given, since the buffer has 20 cells and space for 4 more. In a shared buffer switch element, it doesn’t matter which outputs the cells are addressed to, so long as there is space in the buffer. Similarly, 4 acks will be given in the case of acknowledgement flow control, since there is room for any four arriving cells to be stored.

8. The paper by Choudhury and Hahne (available on the web site) describes a method for regulating the use of the memory space in a shared buffer switch element. Consider the following scenario for a d port switch element with a shared buffer that has space for B cells. At time 0, all input links have a load of 100%, with all the arriving cells directed to output 0. This arrival pattern continues until the queue for output 0 reaches the maximum length allowed by the buffer sharing mechanism. At this point, the arriving traffic changes, so that all arriving cells are directed to output 1. For what values of the dynamic sharing parameter α, will the length of the queue for output 1 match the length of the queue for output 0, before any cells going to output 1 get discarded?

Let \( q_1(t) \) be the length of the first queue at time \( t \). During the initial phase, the queue level rises until \( q_1(t)=aB-q_1(t) \) or equivalently, until \( q_1(t)=(\alpha(\alpha+1))B \). Assume that this happens at time zero, and at this point the traffic pattern changes. Suppose after time zero, \( r>1 \) cells arrive each time period for output 2 (in the most extreme case, \( r=d \)). Then for \( t>0 \), \( q_1(t)=(\alpha(\alpha+1))B-t \) and \( q_2(t)=(r-1)t \). Let \( t_1 \) be the time at which these two become equal, assuming no discards take place. Then, \( t_1=(\alpha(\alpha+1))B(1/r) \). Let \( t_2 \) be the time at which the discarding starts to occur from \( q_2 \). That is, let \( t_2 \) satisfy \( q_2(t_2)=aB-(q_1(t_2)+q_2(t_2)) \) or equivalently when \( (r-1)t_2=aB-((\alpha(\alpha+1))B-t_2+(r-1)t_2) \).

Solving for \( t_2 \) gives \( t_2=(\alpha(\alpha+1))B(1/(r-1)+a(r-2)) \). By setting \( t_1=t_2 \) and solving for \( \alpha \), we get \( \alpha=1/(r-1) \), which means that so long as \( \alpha \leq 1/(r-2) \), the queue lengths will equalize before any discards occur from queue 2. So, for \( r=d \), we need \( \alpha \leq 1/(d-2) \). On the other hand, for \( r=2 \), any value of \( \alpha \) will
satisfy the condition, for \( \alpha = 3 \), any value of \( \alpha \leq 1 \) will satisfy the condition and for \( \alpha = 4 \), any value \( \leq 1/2 \) will satisfy the condition.

9. (15 points) Give the values of \( \tau_{5,6}(12) \) and \( \tau_{9,4}(17) \).

\[
\tau_{5,6}(12) = (12 \mod 5)8 + \left\lfloor \frac{12}{5} \right\rfloor = 16 + 2 = 18
\]
\[
\tau_{9,4}(17) = (17 \mod 9)4 + \left\lfloor \frac{17}{9} \right\rfloor = 32 + 1 = 33
\]

Draw a picture of the network defined by the expression \( X_{2,2} \otimes (X_{2,2} \times X_{3,3}) \otimes X_{2,3} \).

How many paths are there between an input \( x \) of this network, and an output \( y \)?

2

Suppose this network is used to implement a cell switch that uses dynamic routing and balances the load evenly across all available paths in the network. Suppose there is a virtual circuit with a bandwidth of 100 Mb/s from input 4 to output 11. On your diagram, highlight all the links that carry traffic from this virtual circuit and label them with the amount of bandwidth used on each line.
10. Consider the network $K_{n,d,h}$ defined by

$$K_{n,d,0} = D_{n,d} \quad K_{n,d,h} = X_{d,d} \otimes K_{n/d,d,h-1} \otimes X_{d,d} \quad \text{for } h > 0$$

Show that $K_{n,d,h} = D_{n,d,h}^*$ by induction, using the associativity of the parallel connection operation.

Letting $k = \log_d n$,

$$K_{n,d,h} = X_{d,d} \otimes K_{n/d,d,h-1} \otimes X_{d,d}$$

$$\approx X_{d,d} \otimes D_{n/d,d,h-1}^* \otimes X_{d,d}$$

$$= X_{d,d} \otimes (D_{d^{h-1},d}^* \otimes D_{d^{h-2},d}^* \otimes D_{d^{h-1},d}^*) \otimes X_{d,d}$$

$$\approx (X_{d,d} \times D_{d^{h-1},d}) \otimes D_{d^{h-2},d} \otimes (D_{d^{h-1},d} \times X_{d,d})$$

$$= D_{d^h,d}^* \otimes D_{d^{h-2},d} \otimes D_{d^h,d}$$

$$= D_{n,d,h}^*$$

The first step is by definition of $K$, the second step follows by induction, the third step is by definition of $D^*$, the fourth step is by the associativity of the parallel construction operation, the fifth by the associativity of the serial construction operation and the last step by definition of $D^*$. 