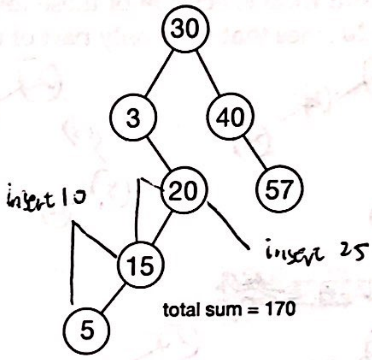
CMU 15-418/618: Parallel Computer Architecture and Programming
Practice Exercise 5

Problem 1: Transactions on Trees

Consider the binary search tree illustrated below.



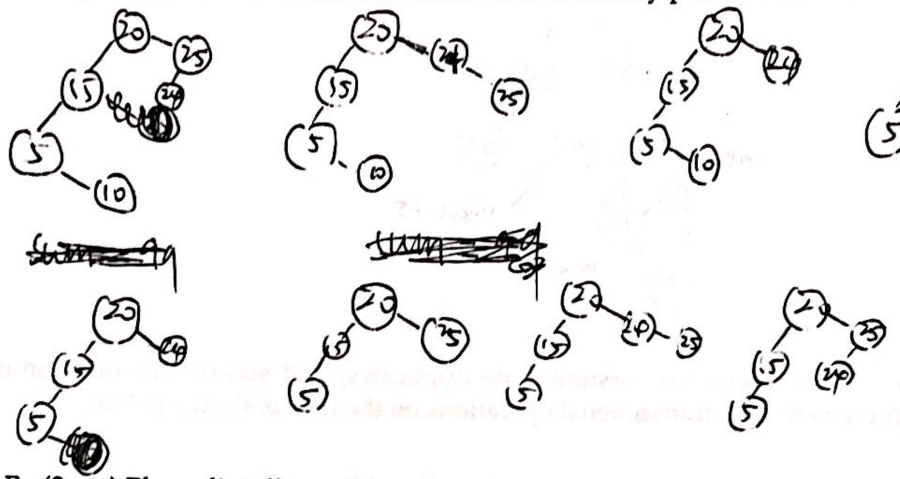
The operations `insert` (insert value into tree, assuming no duplicates) and `sum` (return the sum of all elements in the tree) are implemented as transactional operations on the tree as shown below.

```
struct Node {
   Node *left, *right;
   int value;
};
Node* root;  // root of tree, assume non-null

void insertNode(Node* n, int value) {
  if (value < n) {
    if (n->left == NULL)
       n->left = createNode(value);
    else
       insertNode(n->left, value);
  } else if (value > n) {
    if (n->right == NULL)
       n->right = createNode(value);
    else
       insertNode(n->right, value);
  }  // insert won't be called with a duplicate element, so there's no else case
}

int sumNode(Node* n) {
   if (n == null) return 0;
   int total = n->value;
   total += sumNode(n->left);
   total += sumNode(n->right);
   return total;
}

void insert(int value) {  atomic { insertNode(root, value); }   }
int sum()              {  atomic { return sumNode(root); )     }
```

Consider the following four operations are executed against the tree in parallel by different threads.

```
insert(10);
insert(25);
insert(24);
int x = sum();
```

A. (2 pts) Consider different orderings of how these four operations could be evaluated. Please draw all possible trees that may result from execution of these four transactions. (Note: it's fine to draw only subtrees rooted at node 20 since that's the only part of the tree that's effected.)



B. (2 pts) Please list all possible values that may be returned by `sum()`.

C. (2 pts) Do your answers to parts A or B change depending on whether the implementation of transactions is optimistic or pessimistic? Why or why not?

No, optimistic and pessimistic is just method that implement Tx, they should obey Tx rules. So from outside it makes no different.

D. **(2 pts)** Consider an implementation of **lazy, optimistic** transactional memory that manages transactions at the granularity of tree nodes (the read and writes sets are lists of nodes). Assume that the transaction `insert(10)` commits when `insert(24)` and `insert(25)` are currently at node 20, and `sum()` is at node 40. Which of the four transactions (if any) are aborted? **Please describe why.**

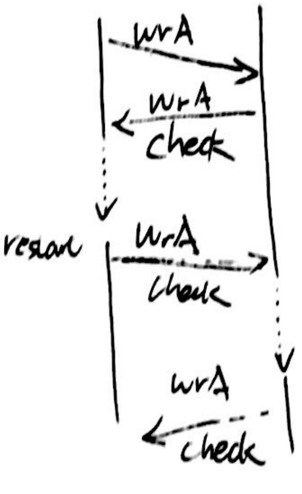
sum() are aborted, because what it reads before is invalidate, since insert(10) only change node 5, so insert(24) and insert(25) still works.

E. **(2 pts)** Assume that the transaction `insert(25)` commits when `insert(10)` is at node 15, `insert(24)` has already modified the tree but not yet committed , and `sum()` is at node 3. Which transactions (if any) are aborted? **Again, please describe why.**

~~insert(24) because insert~~

both insert(24) and insert(10) because insert(25) commits makes node 24 invalidate, then everyone has accessed before that commits needs to access again.

F. **(2 pts)** Now consider a transactional implementation that is **pessimistic** with respect to writes (check for conflict on write) and **optimistic** with respect to reads. The implementation also employs a "writer wins" conflict management scheme – meaning that the transaction issuing a conflicting write will not be aborted (the other conflicting transaction will). Describe how a **livelock problem** could occur in this code.



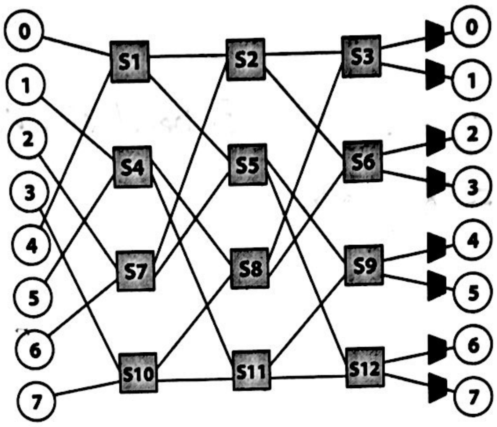
they keep restarting all the time, if they keep check after each restart.

G. (2 pts) Give one livelock avoidance technique that an implementation of a pessimistic transactional memory system might use. You only need to summarize a basic approach, but make sure your answer is clear enough to refer to how you'd schedule the *transactions*.

use randomly - back off , when a thread needs restart it can wait for a range of time, in that case we can avoid from some sence.
Or we can assign priority to the thread whose tid number is greater who got the ticket.

Problem 2: Interconnects

Consider sending two 256-bit packets in the Omega network below. Packet A is sent from node 4 to node 0, and packet B from node 6 to node 1. The network uses worm-hole routing with flits of size 32 bits. All network links can transmit 32 bits in a clock. **The first flit of both packets leaves the respective sending node at the same time. If flits from A and B must contend for a link, flits from packet A always get priority.**



A. (2 pts) What is the latency of transmitting packet A to its destination?

256 / 32 = 8 flits   7 + 4 = 11 cycles.

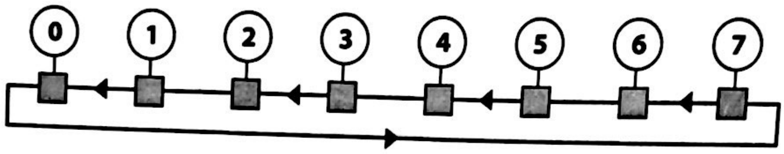
B. (2 pts) What is the latency of transmitting packet B to its destination? Please describe your calculation– switches are numbered to help your explanation.)

A, B get contention in S2, since A has priority.
B has to wait, which is:
wait time is the time when the last flit of Packet reach S2, so wait time should be 5+3=8 cycles.
so Latency should be 8+11 = 19 cycles.

C. (2 pts) If the network used store-and-forward routing, what would be the minimum latency transmitting one packet through the network? (Assume this packet is the only traffic on the network.)

4 cycles

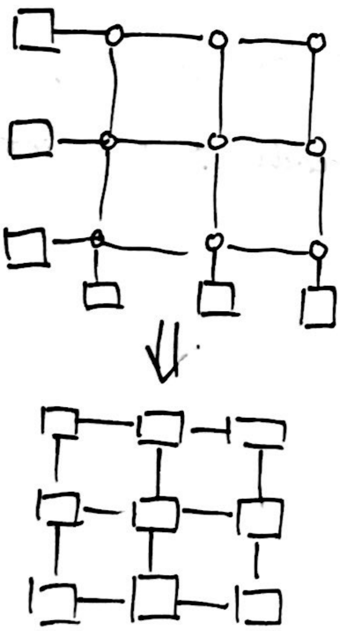
D. (2 pts) Now consider sending packet A from node 2 to node 0 and packet B from node 5 to 3 on the unidirectional ring interconnect shown below. Assuming the conditions from part A (32-bits send over a link per clock, worm-hole routing, same packet and flit sizes, both messages sent at the same time, packet A prioritized over packet B), what is the minimum latency for message A to arrive at its destination? Message B?



A : $7+2 = 9$ cycles.   B need to wait A is send out.

B : $9+7+2 = 18$ cycles.

E. (2 pts) **THIS QUESTION IS UNLRELATED TO THE PREVIOUS ONES.** Consider a parallel version of the 2D grid solver problem from class. The implementation uses a 2D tiled assignment of grid cells to processors. (Recall the grid solver updates all the red cells of the grid based on the value of each cell's neighbors, then all the black cells). Since the grid solver requires communication between processors, you choose to buy a computer with a crossbar interconnect. Your friend observes your purchase and suggests there there is another network topology that would have provided the same performance at a lower cost. What is it? (Why is the performance the same?)



Since processors only send messages to its neighbor, we can put every processor in the 2D grid net, instead of putting them connected with each other.