1. (20 points) Consider the following interference graph:

a. Give a machine with 3 registers, is it possible to find an allocation for the graph above without spilling? You may answer one of yes, no, “I don’t know”. If yes, give a coloring.

b. How does the “improved Chaitin’s algorithm” behave when given this input? Recall that the improved Chaitin’s algorithm keeps variables that may need to be spilled on the “stack”, and tries to assign a register to them in the assignment phase.
2. (20 points) Consider the following control flow graph:

And here is its dominator tree.

What are the natural loops in this flow graph? (Show intermediate steps for partial credit.)
6. (20 points) What is the best software pipelined schedule that you can create for the following precedence graph. This machine has two resources, R0, R1. The edges are labeled by the <iteration difference, latency>; each node is represented by its respective resource reservation table. If an instruction uses resource Rj in cycle i after it has been issued, (i = 0, 1), it is indicated by a black square in row i and column j in the resource reservation table. For example, node 4 uses R0 in cycle 0, and R1 in cycle 1 after the node is issued.

a. What is the bound of the initiation interval?

b. Find the best software pipelined schedule. What is the initiation interval of your schedule? Show the schedule of one iteration.
4. (20 points) Show the points-to result obtained with a flow-insensitive, context-insensitive
inclusion-based points-to analysis. The allocation sites are named \( a1, a2, \ldots, a6. \)

```java
main ()
{
    List list1 = new List(); (a1)
    List list2 = new List(); (a2)
    List list3 = new List(); (a3)
    C elem1 = new C(); (a4)
    D elem2 = new D(); (a5)
    list1.add (elem1);
    list3.add (elem2);
    list2.add (list3);
}
```

```java
public class Node {
    public Node next;
    public Object elem;
}
```

```java
public class List {
    private Node head;
    void add (Object item) {
        Node t = new Node(); (a6)
        t.next = head;
        t.elem = item;
        this.head = t;
    }
}
```

Name the objects pointed to by:

a. \( a1.\text{head} \)

b. \( a2.\text{head} \)

c. \( a3.\text{head} \)

d. \( a6.\text{next} \)

e. \( a6.\text{elem} \)
2. (20 points) Briefly give one important advantage and disadvantage for each of the following styles of garbage collection.

   a. Reference counting
      Advantage:
      Disadvantage:

   b. Copying garbage collection
      Advantage:
      Disadvantage:

   c. Generational garbage collection
      Advantage:
      Disadvantage:

   d. Incremental garbage collection
      Advantage:
      Disadvantage:
5. Assignments in a Java program have implicit type cast operations. For assignment statement

   A x = y;

object y needs to be checked dynamically if it may be type-cast to A. More specifically, the compiled Java bytecode for the above assignment will include a bytecode called CHECKCAST that performs the following operation:

   if (! type(y) is subclass of type(x)) {
       raise exception
   }

a. We can optimize a CHECKCAST bytecode away at compile time if we know that it cannot raise an exception. For the following program, identify all the assignment statements that need no CHECKCAST bytecodes. Assume that class A is a super class of B and B is a super class of C.

   A a1 = x;
   A a2 = x;
   C c1 = x;
   B b1 = x;
b. For the following control flow graph, find all assignments whose `CHECKCAST` bytecode can be optimized away. The class hierarchy is the same as the one above.

![Control Flow Graph]

c. Design a data flow analysis algorithm to remove unnecessary type cast checks. Fill in a table like the one in Problem 4. State clearly which assignments need no `CHECKCAST` bytecodes.