ECE 468
Problem Set 6 Solutions: CFGs and loop optimizations

1. Consider the following code:

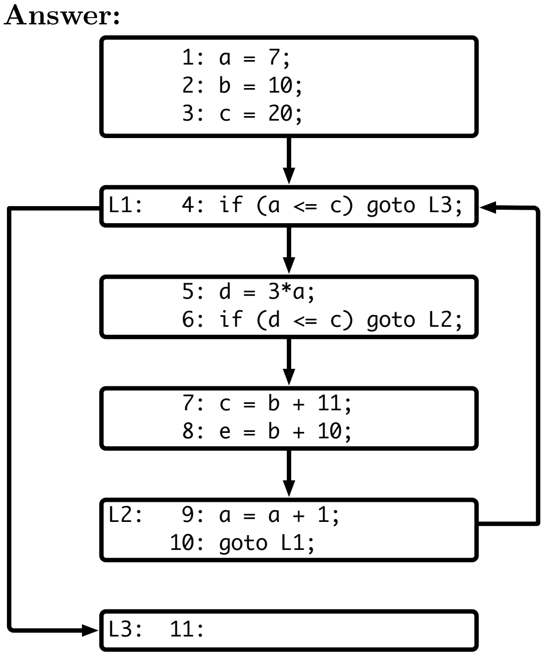
```
        1: a = 7;
        2: b = 10;
        3: c = 20;
L1:     4: if (a <= c) goto L3;
        5: d = 3*a;
        6: if (d <= c) goto L2;
        7: c = b + 11;
        8: e = b + 10;
L2:     9: a = a + 1;
       10: goto L1;
L3:    11: //a, b, c, d and e are all live after this
```

   (a) What are the basic blocks for this code?

   Answer: {1, 2, 3}, {4}, {5, 6}, {7, 8}, {9, 10}, {11}

   (b) Draw the block-level CFG for this code.

   Answer:



(c) What are the loop headers?

Answer: Line 4, because it is the target of a back edge

(d) What are the back edges (give the source and target lines of code)

Answer: The back edge is line 10 to line 4

(e) What are the loop-invariant instructions in this code?

Answer: b does not change in the loop, so lines 7 and 8 are loop invariant.

(f) Which instructions can be moved out of the loop?

Answer: Line 8 can be moved out of the loop, because e is not used anywhere before the loop. We cannot move Line 7 outside the loop, because c is live before the loop. We use c in Lines 4 and 6. If we move Line 7 outside the loop, the first iteration of the loop will use the wrong value of c.

(g) What are the basic induction variables for the loops in this program? The mutual induction variables?

Answer: The basic induction variable is a, and the mutual induction variable is d

(h) Show the code after applying strength reduction

Answer:

```
       1: a = 7;
       2: b = 10;
       3: c = 20;
      3': d' = 3*a;
L1:    4: if (a <= c) goto L3;
       5: d = d';
       6: if (d <= c) goto L2;
       7: c = b + 11;
       8: e = b + 10;
L2:    9: a = a + 1;
      9': d' = d' + 3;
      10: goto L1;
L3:   11: //a, b, c, d and e are all live after this
```

(i) Show the code after applying linear test replacement

Answer:

```
        1: a = 7;
        2: b = 10;
        3: c = 20;
       3': d' = 3*a;
L1:     4: if (d' <= 3*c) goto L3;
        5: d = d';
        6: if (d <= c) goto L2;
        7: c = b + 11;
        8: e = b + 10;
L2:     9: // a = a + 1;
       9': d' = d' + 3;
       10: goto L1;
L3:    11: //a, b, c, d and e are all live after this
```

2. Cam Piler thinks that loop interchange is always a good idea. Prove him wrong in two ways:

   (a) Show that loop interchange is not always correct by giving an example of a nested loop where loop interchange *is not legal*.

   Answer: There are many possible solutions to this problem. In the next set of lectures, we will discuss a general test for whether loop interchange is legal or not. Here is one example where it is not:

   ```
   for (int i = 0; i < N; i++) {
     for (int j = 0; j < N; j++) {
       A[i + 1][j - 1] = 2 * A[i][j];
     }
   }
   ```

   (b) Describe a situation where loop interchange is legal, but is not actually *useful* (and may even be harmful).

   Answer: Again, there are many possible answers here. We are looking for a situation where the existing code has good locality, but the transformed code doesn't. For example, suppose arrays are laid out in row-major order. The following code has good spatial locality:

   ```
   for (int i = 0; i < N; i++) {
     for (int j = 0; j < N; j++) {
       A[i + 1][j + 1] = 2 * A[i][j];
     }
   }
   ```

While the transformed code doesn't:

```
for (int j = 0; j < N; j++) {
  for (int i = 0; i < N; i++) {
    A[i + 1][j + 1] = 2 * A[i][j];
  }
}
```