CSE 401 - Section 10 - Dataflow and Single Static Assignment - Solutions

1. **Dataflow Review** For each of the following optimizations, list the dataflow analysis that would be most directly applicable. You may use a single dataflow analysis for multiple optimizations, or none. The possible dataflow analyses are reproduced here for reference:
   Live Variable Analysis (Determining if there is any path from the definition of a variable to its use along which it is not redefined)
   Reaching Expressions (For an expression, determining which other basic blocks are reached without redefining any of the variables in that expression)
   Reaching Definitions (Determining which other basic blocks could potentially see the value of a given definition)
   Very Busy Expressions (Determining if an expression is evaluated and used along every path that leaves a basic block, and if the value would be consistent in the parent basic block)

a) Constant Propagation - If a variable x is defined to be a constant in one part of the code, replace uses of the variable x with its defined constant.
Reaching Definitions. We need to determine if a definition of a variable (in this case, one that defines it as a constant) is reaching to a use of that variable in order to determine if we can perform constant propagation at that point.

b) Copy Propagation - If a variable x is defined to be equal to a variable y in one part of the code, replace uses of the variable x with the variable y.
Reaching Definitions. Although this problem deals with variables and not constants, it requires the same analysis because it is still looking for the application of a certain definition. Note that this requires slightly more in-depth use of the Reaching Definitions analysis than constant propagation, because it is also necessary to check that there are no other possible definitions of y introduced between the point where x is set to y and the point where copy propagation is to be performed.

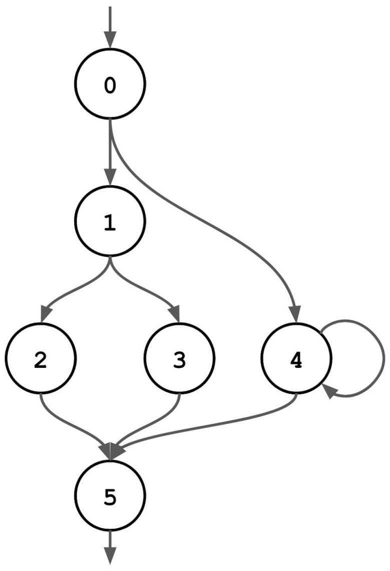
c) Common Subexpression Elimination - If an expression is computed twice and will have the same value in both locations, compute it only once (Note: only applies to expressions without side effects).
Reaching Expressions. The dataflow analysis needs to determine whether a certain expression can be computed only a single time, meaning it should report whether the expression is still valid at a certain point.

d) Code Hoisting - Reducing the size of the code by factoring out duplicate code that appears in all possible paths in a part of the program.
Very Busy Expressions. This dataflow analysis determines if code is computed on all branches emerging from a certain basic block, so it is exactly what is needed for this type of optimization.

e) Dead Store Elimination - Removing assignments to a variable if that assignment will never be used in the program.
Live Variable Analysis. Determining whether a variable will be used after a definition allows us to drop definitions that will never be used. Note that this is distinct from reaching definitions -- in the reaching definitions analysis, we simply determine if a definition is still valid at a certain point, regardless of whether or not it is actually used. We need live variable analysis to see if that definition gets used.

2. **Dominators and Dominance Frontiers** Consider the following simplified control flow graph. For each node in the graph, fill in the table with the set of nodes that are strictly dominated by that node and the set of nodes in its dominance frontier.

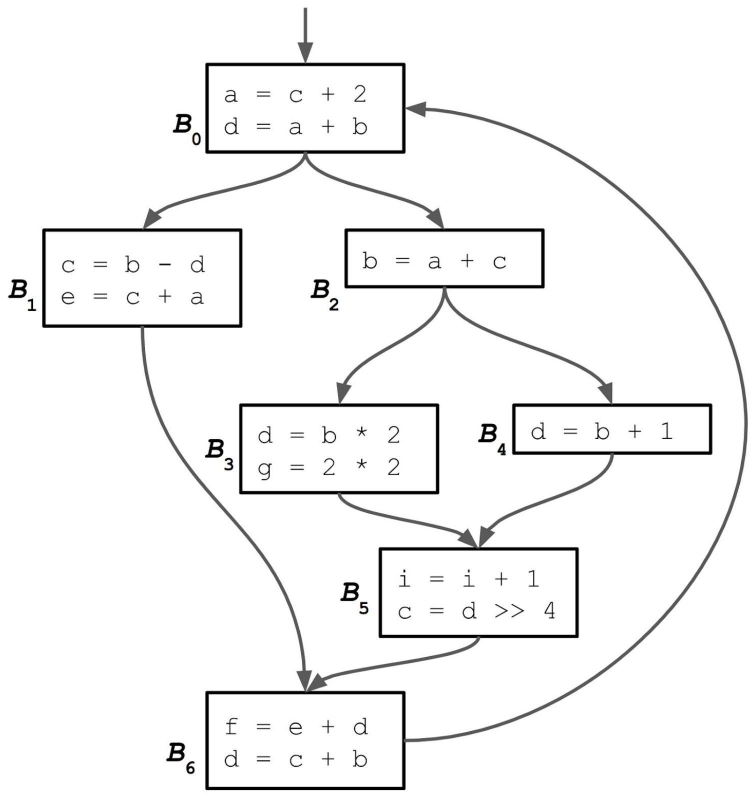
For reference, a node **X** is considered to dominate **Y** iff every path from the entry point of the control flow graph to **Y** includes **X**. A node **X** strictly dominates **Y** if **X** dominates **Y** and **X ≠ Y**. Finally, a node **Y** is in the dominance frontier of node **X** if **X** dominates an immediate predecessor of **Y** but **X** does not strictly dominate **Y**.



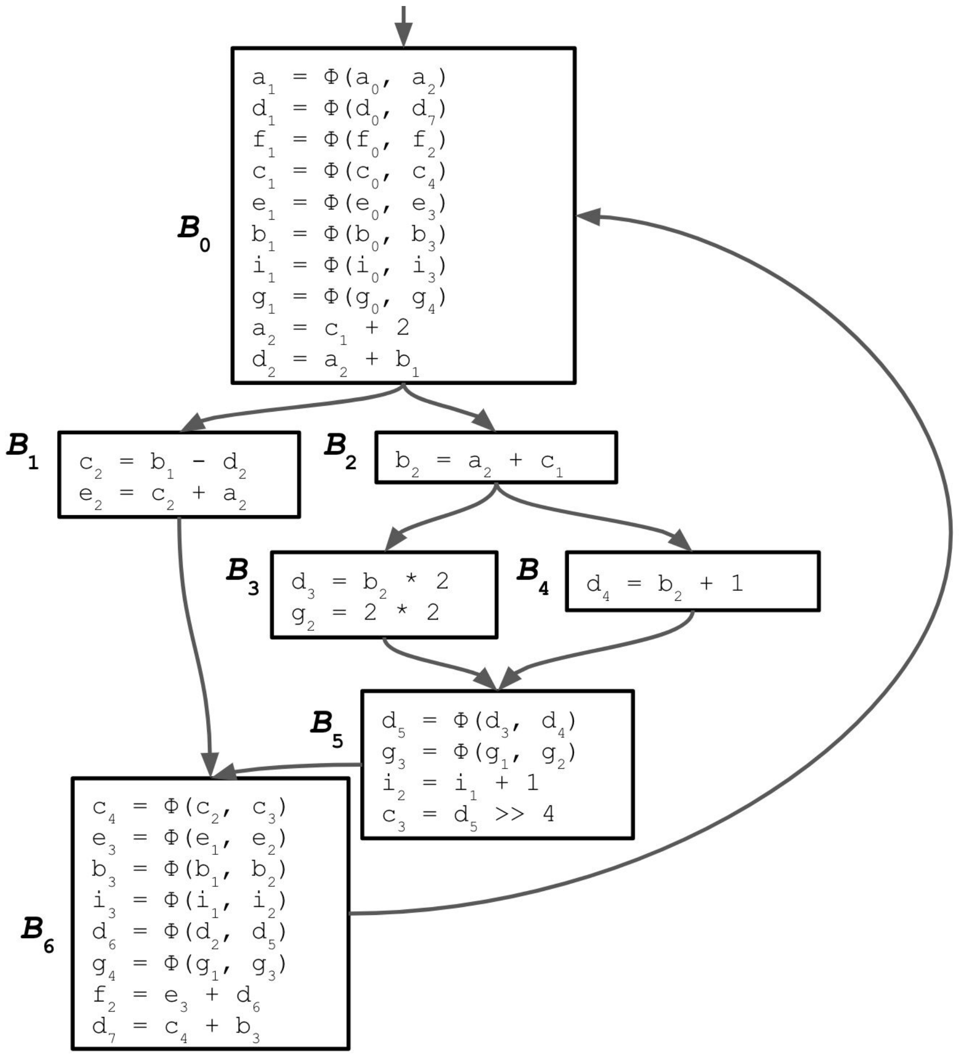
NODE	STRICTLY DOMINATES	DOMINANCE FRONTIER
0	1, 2, 3, 4, 5	∅
1	2, 3	5
2	∅	5
3	∅	5
4	∅	4, 5
5	∅	∅

For a more thorough walkthrough of this problem, see the section 10 slide deck which depicts the relevant sets graphically for each node.

3. **Single Static Assignment** The following is a control flow graph depicting the execution of some arbitrary code. Convert this code to Single Static Assignment form. Hint: you may find it helpful to compute the dominance frontiers of each basic block.



The final SSA-form control flow graph is:



B_0:
$a_1 = \Phi(a_0, a_2)$
$d_1 = \Phi(d_0, d_7)$
$f_1 = \Phi(f_0, f_2)$
$c_1 = \Phi(c_0, c_4)$
$e_1 = \Phi(e_0, e_3)$
$b_1 = \Phi(b_0, b_3)$
$i_1 = \Phi(i_0, i_3)$
$g_1 = \Phi(g_0, g_4)$
$a_2 = c_1 + 2$
$d_2 = a_2 + b_1$

B_1:
$c_2 = b_1 - d_2$
$e_2 = c_2 + a_2$

B_2:
$b_2 = a_2 + c_1$

B_3:
$d_3 = b_2 * 2$
$g_2 = 2 * 2$

B_4:
$d_4 = b_2 + 1$

B_5:
$d_5 = \Phi(d_3, d_4)$
$g_3 = \Phi(g_1, g_2)$
$i_2 = i_1 + 1$
$c_3 = d_5 >> 4$

B_6:
$c_4 = \Phi(c_2, c_3)$
$e_3 = \Phi(e_1, e_2)$
$b_3 = \Phi(b_1, b_2)$
$i_3 = \Phi(i_1, i_2)$
$d_6 = \Phi(d_2, d_5)$
$g_4 = \Phi(g_1, g_3)$
$f_2 = e_3 + d_6$
$d_7 = c_4 + b_3$

The steps taken to compute this final control flow graph are as follows. For a more graphical breakdown of these steps, consult the section 10 slide deck.

1. **The sets of nodes that are strictly dominated by each node and in the dominance frontier of each node need to be calculated. This is exactly the same process as problem 2. Doing this is important because we will use the dominance frontiers to determine where the phi functions need to be placed.**

2. **Initial sets of variables are computed that will be merged at each point. Recall that if a block Y is in the dominance frontier of block X, then any variable which is defined in block X will be given a phi function in block Y (the intuition here is that block Y may have been reached without going through block X, requiring a merge). Therefore, we determine that block 0 needs to merge a, d, and f; block 5 needs to merge d, and g; and block 6 needs to merge c, e, b, and i.**

3. **However, each merge of a variable will itself be a definition storing the result of a phi function. Therefore, if there are variables in the need-to-merge sets for any nodes, they also have to be copied to their dominance frontiers. For example, since d and g are now going to be merged in block 5, and block 6 is in the dominance frontier of block 5, we must add d and g as need-to-merge variables in**

block 6 (ending up with c, e, b, i, d, and g). This repeats until there are no more changes (just once in this example).

4. **Finally, we can produce the SSA code. Every definition of a variable must be distinct, and we simply increment the subscript with each new definition. Phi functions appear at the top of their block. We assume that all variables are defined as “a_0” or “b_0” before entering the control flow graph. If there is a loop backward, it is entirely possible that a definition will have a phi function that contains as an argument a definition with a greater subscript.**