HW week-1
• (a) No. Edge coverage subsumes node coverage. Therefore, any test suite (of any program) that gives full edge coverage will also give full node coverage.

• (b) Let me first note that node coverage does not in general subsume edge coverage. This implies that there exists at least one program P and one test set T that provides full node coverage on P, but T does not give full edge coverage on P. However, on the above program this is actually not possible: any test set that gives full node coverage (on the above program) must also give full edge coverage. Note also that a test-path is required to end in the terminal node. So the answer is “No”.

• (c) Yes. Consider the following test set, consisting of a single test case:
  \{ [1,2,3,2,4] \} \rightarrow it misses the pairs [1,2]-[2,4] and [3,2]-[2,3]

• (d) 2 e.g. the test set: \{ [1,2,4] , [1,2,3,2,3,2,4] \}
(a) TR1 requires us to cover all nodes in G, and TR2 all edges. As discussed in No-1, in general they are not equivalent (they do not subsume each other). The above program demonstrates this. E.g. the test set containing a single test path [1,2,3,2,4,5,6,1,7] covers all nodes, but it misses the edge [4,6].

(d) there are 15 prime paths (pps).
[1,2,4,6,1]  [1,2,4,5,6,1]
[2,3,2]  [2,4,6,1,2]  [2,4,5,6,1,2]
[3,2,3]  [3,2,4,6,1,7]  [3,2,4,5,6,1,7]  → note that reds begin in 3 , and end in terminal 7
[4,6,1,2,4]  [4,5,6,1,2,4]  [4,6,1,2,3]  [4,5,6,1,2,3]  → note that reds begin in 4, and end in non-terminal 3!
[5,6,1,2,4,5]
[6,1,2,4,6]  [6,1,2,4,5,6]

(b) There are 2 pps that start in 1
(c) There are 6 pps that pass 3
No 3

- [0,1,0,2] tours [0,2] (so, it also tours [0,2] with side trip and detour)
- [0,1,2] does not tour [0,2], not even with side trip.
- [0,1,2] tours [0,2] with detour
- [0,1,2,0,1,0,2] does not tour[2,0,2]. However, it does tour [2,0,2] with side trip (so, also with detour).
HW week-2
(a) The test set \{ (0,1,2), (-1,-1,-1) \} does not give full ECC. For example, it misses blocks b1 and b2 of Side-1, and blocks b1 and b3 of Side-2.

(b) The test set \{ (0,1,2), (2,0,1), (1,2,0), (-1,-1,-1) \} does give full ECC.
Table 4.2. Second partitioning of TriTyp's inputs (interface-based).

<table>
<thead>
<tr>
<th>Partition</th>
<th>$b_1$</th>
<th>$b_2$</th>
<th>$b_3$</th>
<th>$b_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_1 = \text{&quot;Length of Side 1&quot;}$</td>
<td>greater than 1</td>
<td>equal to 1</td>
<td>equal to 0</td>
<td>less than 0</td>
</tr>
<tr>
<td>$q_2 = \text{&quot;Length of Side 2&quot;}$</td>
<td>greater than 1</td>
<td>equal to 1</td>
<td>equal to 0</td>
<td>less than 0</td>
</tr>
<tr>
<td>$q_3 = \text{&quot;Length of Side 3&quot;}$</td>
<td>greater than 1</td>
<td>equal to 1</td>
<td>equal to 0</td>
<td>less than 0</td>
</tr>
</tbody>
</table>

(c) The following 16 test cases give full PWC:

- (2,2,2) (2,1,-1) (2,0,0) (2,-1,1)
- (1,2,1) (1,1,2) (1,0,-1) (1,-1,0)
- (0,2,0) (0,1,1) (0,0,2) (0,-1,-1)
- (-1,2,-1) (-1,1,0) (-1,0,1) (-1,-1,2)

(d) I manage it with 16, let me know if you can come up with less.
No 2

Consider again the above partitioning of TriTyp. This time we want to apply MBCC, using the following choice of base blocks: \{b1, b2\} for q1, \{b1\} for q2, and \{b1\} for q3 (for your convenience: b1 represents greater than 1, and b2 represents exactly 1).

- Consider the following test set T. It gives full MBCC. Furthermore, it is minimal (the smallest test set that gives MBCC).
  - Base tests: (2,2,2) and (1,2,2)
  - Additionals:
    - wrt q1: (0,2,2) and (0,2,2) \rightarrow dup
      - (1,2,2) and (-1,2,2) \rightarrow dup
    - wrt q2: (2,1,2) and (1,1,2)
      - (2,0,2) and (1,0,2)
      - (2,-1,2) and (1,-1,2)
    - wrt q3: (2,2,1) and (1,2,1)
      - (2,2,0) and (1,2,0)
      - (2,2,-1) and (1,2,-1)
  - Because of two duplicates, effectively you only have 16 test cases.

- (a) Yes, MBCC subsumes ECC.
- (b) No.
- (c) MBCC does not require you to cover combinations among non-base blocks. Since T is assumed to be minimal, it won’t cover such non-base blocks combinations, including pairs over them. For example, the pairs (b3,b3,x) or (b3,x,b3) will be missed by T due to its minimality towards MBCC. So … the answer is “No”. Note that a test set T’ that gives full MBBC but is not minimal might give full PWC. A trivial example is if T’ simply includes all combinations over all blocks. MBBC may not give full PWC. The above test set is an example of the latter case (it does not give you full PWC).
- (d) According to the formula given in the text, we should get at most 2(base) + 4 + 6 + 6 = 18. Two of these are duplicates (as pointed out above), so the result is 16.
HW week-3
• (a) *Is it possible that a program contains a prime path which is not a du-path of any variable?*
   Yes. Example: `getx() { return x }`, which does not define any variable, and hence has no du-path.

• (b) *Suppose a program P contains a variable x. Is it possible that it contains a du-path for x which is not a simple path?* No. Any du-path is by definition a simple path.

• (c) *Suppose [n1,n2,n3] is a du-path for x. Where do writes to x take place?* In any case in the node n1. By definition, n3 will contain a use of x. n3 may also contain a def of x, but if this is the case we assume this def to occur, in the code fragment represented by n3, after the use of x in n3.
(a) Let us first list down all the du-paths of x. See below. I organize them in def-pair sets, and I will also give names to the paths so that I can refer to them later:

\[ du(0,7,x) = \{ \text{dp1} \} \text{ where } \text{dp1 is the path } [0, 1, 7] \]
\[ du(0,5,x) = \{ \text{dp2} \} \text{ where } \text{dp2 is the path } [0, 1, 2, 4, 5] \]
\[ du(3,5,x) = \{ \text{dp3} \} \text{ where } \text{dp3 is } [3, 2, 4, 5] \]
\[ du(3,7,x) = \{ \text{dp4,dp5} \} \text{ where } \text{dp4 is } [3, 2, 4, 6, 1, 7], \text{ and } \text{dp5 is } [3, 2, 4, 5, 6, 1, 7] \]

So... in total there are 5 du-paths of x.

(b) Roughly said, full \textit{all-defs} coverage over x requires you to cover all \textquotedblleft def\textquotedblright-locations of x. More precisely, you are require to cover all (non-empty) def-path sets of x. These are all x's def-path sets:

\[ du(0,x) = \{ \text{dp1,dp2} \} \]
\[ du(3,x) = \{ \text{dp3,dp4,dp5} \} \]

Notice that \[ du(0,x) = du(0,5,x) \cup du(0,7,x) \], and \[ du(3,x) = du(3,5,x) \cup du(3,7,x) \]. The TR for \textit{full all-defs} coverage over x consists of one chosen path from each of the above def-path sets. For example, this TR could be \{dp1,dp3\}. But TR = \{dp2,dp5\} is also good. Regardless the choice, we have two def-path sets, so the TR will always contain two paths. (so the final answer is \textquotedblleft2\textquotedblright).

(c) Roughly, full \textit{all-uses} coverage over x requires you to cover all pairs of def and use locations of x. More precisely, you need to cover all (non-empty) def-pair sets as listed down in (a). The TR will consists of one chosen path from each def-pair. For example it could be TR=\{dp1,dp2,dp3,dp4\}. But it could also be TR=\{dp1,dp2,dp3,dp5\}. In this case, this is the only two possible TRs. The size of this TR is 4 (because there are 4x def-pairs of x). So the final answer is also \textquotedblleft4\textquotedblright.
Now we can easily answer each question:

(a) There are 7 du-paths for x, 2 for y, and 2 for a.
(b) To get all du-path coverage for y you need to cover all du-paths for y. There are 2.
(c) To get full all-uses coverage for a you need to cover all du-pairs of a. You end up with two paths to cover. See also the explanation of no-2.
(d) 7. See also the explanation of no 2.
HW week-4
The data flows of `stut()` and `checkDups()`

The picture below shows again the control flow graphs of the two programs, `sust` and `checkDups`, decorated with the def and use information of variables involved. There are two call sites of interest, marked blue in the picture. The **coupling vars** are: `linecnt/line, lastdelimit, curWord`. We will first list all the coupling paths for each of these variables, for each call site.

0: def = `{inFile, inLine, c, linecnt}`
1: use = `{inFile}`
   def = `{inLine}`
2: use = `{i, inLine}
   def = `{i}`
3: use = `{inLine, i}
   def = `{c}`
4: use = `{c}
   def = `{linecnt}`
5: use = `{linecnt}
   call checkDups(linecnt)
6: use = `{linecnt}
   def = `{linecnt}`
7: use = `{curWord, c}
   def = `{lastdelimit, curWord}`
8: use = `{linecnt}
   call checkDups(linecnt)
9: exit

0: def = `{line} => just param passing
1: use = `{lastdelimit}
2: exit
3: def = `{lastdelimit}
4: use = `{curWord, prevWord}
5: use = `{curWord, prevWord}
6: use = `{curWord}
   def = `{prevWord}
7: def = `{curWord}
8: exit
The *coupling vars* are `linecnt/line`, `lastdelimit`, `curWord`. Nodes where we have either a last def or first use of these variables, in either side of the call, have been marked green above, just to help you locating them. I will write s0, s1 etc to mean the nodes of “stut”, and c0, c1 etc to mean the nodes of checkDups.

The coupling paths of these variables for the call site at location s8 are listed below. Do note that a couple path is a du-path, and therefore must also be a simple path.

- Since `linecnt` is just pass through as the parameter `line` to checkDups, for deciding first-uses at checkDups I treat them as the same. However, this parameter is passed by value, so the parameter `line` is not a coupling-var towards `stut`.
  - `linecnt` has only one coupling path: [s6, s8, c0, c1, c3, c4, c5]
  - `lastdelimit` also has one coupling path: [s7, s2, s6, s8, c0, c1]
  - `curWord` has two coupling paths: [s7, s2, s6, s8, c0, c0, c1, c3, c4], [c7, c8, s8, s1, s2, s3, s4, s7]
The coupling vars are linecnt/line, lastdelimit, curWord. The coupling paths of these variables for the call site at location s5 are listed below:

- linecnt has in any case this coupling path: [s0,s1,s2,s3,s4,s5,c0,c1,c3,c4,c5]. Additionally, there is a def of linecnt at s6. The information of this def flows however through the call site s8. However, since we know the call does not affect linecnt, we will also count [s6,s8,s1,s2,s3,s4,s5, c0,c1,c3,c4,c5] as a coupling path for linecnt.
- lastdelimit also has one coupling path: [s7,s2,s3,s4,s5, c0,c1]
- curword has two coupling paths: [s7,s2,s3,s4,s5,c0,c0,c1,c3,c4], [c7,c8,s5,s2,s3,s4,s7]
Now we can answer the questions:

(a) For call site s8, we have:
   - linecnt: 1 coupling path
   - lastdelimit: 1 coupling path
   - curword: 2 coupling paths

(b) For call site s5, we have:
   - linecnt: 2 coupling paths
   - lastdelimit: 1 coupling path
   - curword: 2 coupling paths

(c) Well, for each call site, we only have one coupling path for lastdelimit. So, to get full "All Coupling Uses Coverage" (ACUC) over lastdelimit, these are the only paths need to be covered. So, the final answer is “1 per call site”.

(d) For each call site, curword has two coupling paths. Notice that each has different last-def locations. So, for full "All Coupling Uses Coverage" we will have to cover them all. So, the final answer is “2 per call site”. Btw, notice that even for the weaker “All Coupling Defs Coverage” we have to cover both coupling paths of curword (per call site).