Test Data for Comparing Code

Algorithms:

This problem can be broken into two pieces: determining if there exists a mapping that takes a particular set of lines from the first program to a particular set of lines from the second (call this the inner algorithm), and determining the best such range (call this the outer algorithm). The efficiency of the outer algorithm depends on the functions provided by the inner one.

Given two sets of lines, you can compute if there is a variable mapping taking the entire first set to the other fairly efficiently. Each line is an equation. The corresponding equation of an equation is the equation on the same line in the other program. For each variable mentioned in each program, look at all the lines in which it appears in that program. If it ever appears on the left-hand side, then the variable must be mapped to the value on the left-hand side in the corresponding equation in the other program. If it appears on the right-hand side twice in the some equation, it must match to the value on the right-hand side in the corresponding equation in the other program (and the corresponding equation must use the same variable). If a variable appears only on the right-hand sides of equations in one program, then the corresponding right-hand sides of all occurrences of that variable must share a common variable. Clearly, if a variable must map to two different variables, then no mapping exists. If the variable mapping of some variable X is known to be Y, then if any occurrence of X does not have Y in the corresponding equation, then no mapping exists. Note that variables from both programs must be tested.

All of these conditions are trivially necessary, but it may be less clear that they are sufficient. If a variable X must map to another variable Y by the conditions above, then it is checked by the conditions above to ensure that every occurrence of variable X can be mapped to the variable Y. Note that variable X is known to map to variable Y if and only if variable Y is known to map to variable X. If variable X can match to either variable Y or to variable Z ($Y \neq Z$), then there are several situations: the mapping for neither variable Y nor variable Z is known, the mapping for either variable Y or variable Z is known, or the mappings for both variable Y and variable Z are known.

Neither known: If neither mapping is known, then every time variables Y or Z appear, they must appear on the right-hand side of an equation. Furthermore, the right-hand side of the corresponding equation must be X and some other variable W. Thus, the variables can map either way.

One known: If variable X can map to either Y or Z, then there is some line that causes variable X and some variable W to possibly map to either Y or Z. Without loss of generality, presume the mapping of Y is known. Since Y cannot map to X, it must map to W, or the conditions would have rejected the program set. Since the mapping of variable Z is not known, then it must not appear in any other places or always appear paired with X and W. Thus, X can be mapped to Z, and vice-versa.

Both known: If variable X can map to either Y or Z, then there is some line that causes variable X and some variable W to possibly map to either Y or Z. If the mapping of Y and Z are known, then they must map to that variable W, which is both not legal and will be caught by the conditions above.

For the purposes of analysis, assume, without loss of generality, $R \leq H$.

Algorithm 1: For each line offset from –R to R, start at the first pair of lines that differ by the desired offset. Attempt to add pairs of lines to the end of each subprogram. If a pair of lines cannot be added (as determined an incremental algorithm that tests the conditions above), delete pairs of lines from the start of the subprograms until the pair of lines can be added or until the subprograms become empty.

This algorithm is O(R H), presuming the conditions are incrementally checked in O(1) time. It requires a method to incrementally update the conditions both by adding lines to the end of the subprograms of consideration and delete lines from the beginning of the subprograms.

This algorithm is expected to receive full points.

Algorithm 2: Start at a pair of lines from each program. Attempt to add pairs of lines to the end of each subprogram until the addition causes a conflict.

This algorithm is $O(R^2 H)$, presuming the conditions are incrementally checked in O(1) time. It requires a method to incrementally update the conditions both by adding lines to the end of the subprograms of consideration.

This algorithm is expected to receive about 65% of the points.

Algorithm 3: For each pair of starting locations, consider each possible program pairing with that pair of starting location, doing binary search to find the optimal length.

This algorithm is $O(R^2 H \log R)$ time, presuming the time to check the existence of a mapping can be done in time linear to the number of lines. It does not require support for any incremental operations.

This algorithm is expected to receive about 55% of the points.

Algorithm 4: Rather than do the O(1) mapping checker, use maximum-weighted bipartite matching. Add an edge between every pair of variables names with weight equal to the number of pairings satisfied by their pairing. If the maximum weight bipartite matching

is 3·k, where k is the number of lines in the subprograms of consideration, then there exists a mapping.

The efficiency of this algorithm varies based on the outer algorithm used. This algorithm is expected to receive about 45% of the points, although this will vary depending on algorithm and implementation.

Algorithm 5: Attempt to construct a variable mapping by constrained search of all possible mappings.

The efficiency of this algorithm varies dramatically based on the exact algorithm used. In general, this class of algorithms is expected to receive about 30% of the points.

Test Data

Test #	Points	R	H	Answer
1	5	4	4	4
2	5	5	4	4
3	5	9	6	0
4	5	5	4	4
5	5	12	3	3
6	5	10	1	1
7	5	200	10	2
8	5	90	70	43
9	5	125	110	40
10	5	180	170	60
11	5	230	210	132
12	5	354	318	178
13	5	403	364	198
14	5	465	438	203
15	5	523	497	211
16	5	678	659	212
17	5	804	787	256
18	5	904	909	274
19	5	1000	1000	306
20	5	1000	1000	98