Massachusetts Institute of Technology | Handout 17
6.042J/18.062J: Mathematics for Computer Science | March 3, 2000
Professors David Karger and Nancy Lynch

Problem Set 4.5 Solutions

Problems:

Problem 1 This problem is repeated from last Friday's tutorial, slightly rewritten.

Math Moose has been given a new puzzle: a 3×3 grid of squares, where each square contains one of the digits in the set $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$, each digit appearing exactly once in the puzzle. Each row can be rotated right or left and each column can be rotated up or down. The effects of such rotations are demonstrated in the following figure.

$$\begin{array}{|c|c|c|}\hline 2 & 3 & 6 \\ \hline 7 & 8 & 9 \\ \hline 5 & 4 & 1 \\ \hline \end{array} \rightarrow \begin{array}{|c|c|c|}\hline 5 & 3 & 6 \\ \hline 2 & 8 & 9 \\ \hline 7 & 4 & 1 \\ \hline \end{array} \rightarrow \begin{array}{|c|c|c|}\hline 5 & 3 & 6 \\ \hline 8 & 9 & 2 \\ \hline 7 & 4 & 1 \\ \hline \end{array}$$

Figure 1: Examples of puzzle row and column rotations.

The first picture shows the initial configuration of the puzzle. The leftmost column is then rotated downward to give the middle picture. The 2 and 7 move down and the 5 rotates back up to the top. Then, the middle row is rotated leftward to produce the final picture. The 8 and 9 move left and the 2 rotates back around to the right.

Since discovering the puzzle, Math Moose has spent hours trying to obtain the following configuration, starting of course from the initial configuration of Figure 1 and using legal row and column rotations.

1	2	3
4	5	6
7	8	9

Sad to say, Math Moose has failed miserably. He is beginning to suspect that the puzzle might be designed in a way that makes this configuration impossible, no matter what sequence of row and column rotations one attempts.

(a) Model the puzzle as a state machine.

Solution.

The puzzle can be modeled by the state machine M defined as follows:

- The set of states Q consists of all arrangements of the digits in the set $\{1,2,3,4,5,6,7,8,9\}$ on a 3×3 grid where each digit appears exactly once.
- The set of start states Q_0 consists of the single arrangement 236 789 541.
- The relation $\delta \subseteq Q \times Q$ consists of all pairs of states (q_1, q_2) such that q_2 can be obtained from q_1 by a row or column rotation.

□

(b) Given the initial configuration of the puzzle indicated above, namely 236 789 541, prove that Math Moose can never obtain the configuration 123 456 789.

Solution.

The idea is to keep track of the number of digit pairs that are out of order relative to the desired configuration 123 456 789. For example, in the initial configuration, 2 appears before 1; this pair of digits is out of order, because 1 appears before 2 in the desired configuration 123 456 789. We will refer to an out-of-order digit pair as an *inversion*. In the initial configuration, there are 17 inversions:

21	31	65	64	61
75	74	71	85	84
81	95	94	91	54
51	41			

The key observation is that a rotation always changes the number of inversions by an even amount. Since the puzzle initially has an odd number of inversions, no sequence of rotations can reduce the number of inversions to 0, an even number. Consequently, there will always be at least one pair of digits out of order.

The main challenge is to prove that a rotation always changes the number of inversions by an even amount. We will prove this claim in a sequence of lemmas, and then we will prove the main result as a theorem.

Lemma 1.1. *Swapping an adjacent pair of digits changes the number of inversions by an odd amount.*

Proof. If the digits were initially in the same order as in 123 456 789, then the number of inversions is increased by 1, an odd number. Otherwise, the digits were initially in the wrong order, and the number of inversions is decreased by 1, also an odd number. ■

Lemma 1.2. *Swapping any pair of digits changes the number of inversions by an odd amount.*

Proof. We will show how to swap any pair of digits using an odd number of swaps of adjacent digits. The preceding lemma says that swapping adjacent digits changes the number of inversions by an odd amount. Therefore, swapping any pair of digits changes the number of inversions by an odd amount, because the sum of an odd number of odd numbers is odd.

All that remains is to show how to swap any pair of digits using an odd number of swaps of adjacent digits. Suppose that we are swapping digits x and y, where x is on the left, y is on the right, and there are k stray digits between the two. First, we swap x with its right neighbor k times. This leaves x directly left of y and leaves all the stray digits shifted one place to the left. Then we swap x and y, putting x into y's original positions. Finally, we swap y with its left neighbor k times. This puts y into x's original position and shifts the stray digits back to their original position. The total number of swaps of adjacent digits is $k + 1 + k = 2k + 1$, which is odd for all $k \geq 0$. ∎

Lemma 1.3. *Rotating any row or column changes the number of inversions by an even amount.*

Proof. We show below that any row or column rotation can be obtained by two swaps. The preceding lemma says that every swap changes the number of inversions by an even amount. Since the sum of two odd numbers is even, any row or column rotation changes the number of inversions by an even amount.

Here is how to obtain a rotation from two swaps. Suppose that the digit sequence xyz appears in some row or column. Swapping x with y and then swapping x with z creates the sequence yzx, which corresponds to a leftward or upward rotation. Swapping z with y and then z with x creates the sequence zxy. This corresponds to a rightward or downward rotation. ∎

Now define $P(q)$ to be the predicate saying that, in state q of M, the number of inversions in the digit sequence relative to the 123 456 789 sequence is odd. Let $P'(n)$, for $n \geq 0$, be the predicate that asserts P for all states that can be reached by n-step executions of M.

Theorem 1.4. *$P'(n)$ is true for all $n \geq 0$.*

Proof. By induction on n, the length of the execution.

For the base case, we need to show that the property holds in the start state, *i.e.*, an execution of length $n = 0$. This is true because in the initial configuration, the number of inversions is odd, in particular equal to 17. For the inductive step, we must show that if P holds for any configuration reachable after n steps of M, then P holds for any configuration reachable after $n+1$ steps of M. Fix $n \geq 0$ and let $q_0, q_1, \ldots, q_n, q_{n+1}$ denote any $n+1$-step execution of M. q_n is thus the final state of an n-step execution of M. Thus, by our inductive hypothesis, in the state q_n, the puzzle has an odd number of inversions. Now, consider the final step (q_n, q_{n+1}) of the $n+1$-step execution of M. By Lemma 1.3, this step changes the number of rotations by an even amount. It follows that the number of rotations of the state q_{n+1} is odd. Thus, by induction, we have shown that P is an invariant property of M. ∎

Corollary 1.5. *Math Moose cannot obtain the configuration 123 456 789.*

Proof. This has an even number (0) of inversions. ∎

Problem 2 A robot wanders on a 2-dimensional grid, starting at the origin, $(0, 0)$. In one step, he is able to move unit distance along any diagonal line, that is, at an angle of ± 45 degrees or ± 135 degrees.

(a) Describe how to model the robot's behavior as a state machine.

Solution.

The possible states are: $Q = \{(x, y)\} \subset (\mathbb{R} \times \mathbb{R})$

The initial state is: $Q_0 = (0, 0)$

The transitions are: $\delta \subseteq Q \times Q = \{(q1, q2)$ such that $q2$ is ± 45 degrees or ± 135 degrees from $q1\}$

$\square$

(b) Give an execution of the state machine that reaches the point $(\frac{5\sqrt{2}}{2}, \frac{-3\sqrt{2}}{2})$.

Solution.

Starting at Q_0 make the following transitions:

1. -45 degrees: New state is $(\frac{\sqrt{2}}{2}, \frac{-\sqrt{2}}{2})$
2. -45 degrees: New state is $(\frac{2\sqrt{2}}{2}, \frac{-2\sqrt{2}}{2})$
3. -45 degrees: New state is $(\frac{3\sqrt{2}}{2}, \frac{-3\sqrt{2}}{2})$
4. 45 degrees: New state is $(\frac{4\sqrt{2}}{2}, \frac{-2\sqrt{2}}{2})$
5. -45 degrees: New state is $(\frac{5\sqrt{2}}{2}, \frac{-3\sqrt{2}}{2})$

$\square$

(c) Prove (using induction on the length of the execution) that the robot cannot reach the point $(5, 5)$.

Solution.

We will prove a stronger theorem first and then apply it to our special case.

Theorem. *After any n step execution (starting at Q_0), the x and y coordinates of the final state are both multiples of $\frac{\sqrt{2}}{2}$.*

Proof. Let $P(n)$ be the predicate that after n steps, the x and the y coordinates of the current state are both multiples of $\frac{\sqrt{2}}{2}$.

Base Case: $P(0)$ is true since $Q_0 = (0, 0)$ and 0 is a multiple of $\frac{\sqrt{2}}{2}$.

Inductive Step: We must show that for any positive integer n, $P(n) \implies P(n+1)$. We can do this by picking an arbitrary n step execution and enumerating all the possibilities for the $(n+1)$th step. By the inductive hypothesis, after our n step execution, the current x and y coordinates are both multiples of $\frac{\sqrt{2}}{2}$. Therefore, we can pick integers k and j and call our current state: $(\frac{k\sqrt{2}}{2}, \frac{j\sqrt{2}}{2})$

Now there are 4 possible transitions to the new state, Q_{n+1}:

$$\begin{array}{rcccc} 45\ degrees: & Q_{n+1} & = & (\frac{k\sqrt{2}}{2}+\frac{\sqrt{2}}{2}, \frac{j\sqrt{2}}{2}+\frac{\sqrt{2}}{2}) & = & (\frac{(k+1)\sqrt{2}}{2}, \frac{(j+1)\sqrt{2}}{2}) \\ -45\ degrees: & Q_{n+1} & = & (\frac{k\sqrt{2}}{2}+\frac{\sqrt{2}}{2}, \frac{j\sqrt{2}}{2}-\frac{\sqrt{2}}{2}) & = & (\frac{(k+1)\sqrt{2}}{2}, \frac{(j-1)\sqrt{2}}{2}) \\ 135\ degrees: & Q_{n+1} & = & (\frac{k\sqrt{2}}{2}-\frac{\sqrt{2}}{2}, \frac{j\sqrt{2}}{2}+\frac{\sqrt{2}}{2}) & = & (\frac{(k-1)\sqrt{2}}{2}, \frac{(j+1)\sqrt{2}}{2}) \\ -135\ degrees: & Q_{n+1} & = & (\frac{k\sqrt{2}}{2}-\frac{\sqrt{2}}{2}, \frac{j\sqrt{2}}{2}-\frac{\sqrt{2}}{2}) & = & (\frac{(k-1)\sqrt{2}}{2}, \frac{(j-1)\sqrt{2}}{2}) \end{array}$$

In all four situations, our resulting state still satisfies the predicate. Therefore, we have $P(n) \implies P(n+1)$.

Finally, since the x and y coordinates of any reachable state are always multiples of $\frac{\sqrt{2}}{2}$, $(5,5)$ is not reachable.

■

Problem 3 Suppose that the following procedure is used to raise a real number x to the non-negative integral power y.

- Do the assignments $a := x, b := y, p := 1$.
- Repeat the following steps:
  - If b is 0, output p and terminate.
  - If b is even, do the assignment $a := a^2$ (that is, set the next value of a to be the square of the current value), and also do $b := b/2$.
  - If b is odd, do $p := p \cdot a$ and $b := b - 1$.

(a) Model the procedure with a state machine.

Solution.
The state can be represented with a triple of numbers a, b, p:

$$Q = \{a, b, p\} \subset (\mathbb{R} \times \mathbb{N} \times \mathbb{R})$$

The initial state is

$$q_0 = (x, y, 1)$$

There are two transitions:

$$\begin{array}{rcl} (a,b,p) & \rightarrow & (a^2, b/2, p) \text{ if } b \text{ is even and } b > 0 \\ (a,b,p) & \rightarrow & (a, b-1, ap) \text{ if } b \text{ is odd} \end{array}$$

The terminating condition is $b = 0$, and then the state machine outputs p.

(b) Formulate an invariant relating a, b, x, y, and p that holds in every state. Your invariant should imply that if $b = 0$, then $p = x^y$.

Solution.

The invariant is expressed by the following predicate $P(\cdot)$ which takes a state of the machine as an argument:

$$P(a, b, p) = \text{“ } a^b p = x^y \text{ ”}$$

Note that if $b = 0$ and $P(a, b, p)$ then $a^b p = p = x^y$.

□

(c) Prove that the invariant holds.

Solution. We use the Invariant Theorem. Clearly, $P(q_0)$ holds:

$$P(q_0) = \text{“ } (x^y) * 1 = x^y \text{ ”}$$

Now, we show that if $P(q)$ and (q, q') is a valid transition then $P(q')$:

Take some $q = (a, b, p)$ and assume $P(q)$, i.e. that $a^b p = x^y$. $q' = (a', b', p')$ could be a state following q if one of the two cases holds:

- If $b > 0$ and b even then $(a', b', p') = (a^2, b/2, p)$, and then

  $$(a')^{b'} p' = (a^2)^{(b/2)} p = a^b p = x^y$$

  where the last equation follows from $P(q)$. Hence, in this case $P(q')$ is true.

- If b odd then $(a', b', p') = (a, b-1, ap)$, and then

  $$(a')^{b'} p' = a^{b-1} ap = a^b p = x^y$$

  where again, the last equation follows from $P(q)$. Hence, in this case $P(q')$ is true as well.

□

(d) Prove that the procedure terminates.

Solution.

We need to show that in any live execution, b eventually reaches 0, because that's the termination condition. Let's argue the following facts:

1. It is an invariant of this state machine that b is an integer. It starts as an integer y, and the two transitions preserve that: If $b \in \mathbb{Z}$ and b is even then $b/2 \in \mathbb{Z}$. Similarly, if $b \in \mathbb{Z}$ then $(b-1) \in \mathbb{Z}$.

2. b cannot be a negative number. We can show that by a contradiction: If b ever reached a negative number, since b starts from a nonnegative integer y, there must be a transition where b went below 0, i.e. a transition from $b \geq 0$ to $b' < 0$. But that's impossible with our transitions:
   - If $b \geq 0$ then $b/2 \not< 0$
   - If $b \geq 0$ and $b \neq 0$ then $b - 1 \not< 0$.
3. b decreases in every transition (obviously, if $b > 0$ than $b/2 < b$ and $b - 1 < b$).

Facts (1), (2) and (3) imply that b must attain the value $b = 0$ and trigger the state machine to terminate.

Formally, it the the well-ordering principle that we use here: From facts (1) and (2) it follows that for any execution of this state machine, the set of values of variable b is a subset of $\mathbb{N}$. Therefore, from the well-ordering principle, it must have the smallest element b_0.

We claim that b_0 must be 0. If not, then since the execution is live, another step must follow, which (by fact (3)) implies that the next value of b must be smaller. But we have assumed that b_0 is the smallest value attained by b. This is a contradiction, which implies that $b_0 = 0$.

Therefore, the procedure must terminate.

$\square$

Problem 4 The nondeterministic algorithm $NSort$ starts with a sequence S of n distinct integers, initially in any order. $NSort$ describes how to sort the sequence, in place, into increasing order. What it does is to choose, arbitrarily, any pair of adjacent numbers that appear in the wrong order, that is, that form an "inversion", as in Problem . It corrects this inversion by interchanging these two numbers, leaving the rest of the sequence S unchanged. $NSort$ repeats this step until no further steps can be applied.

(a) Model this as a state machine.

Solution.

The state is represented by a string:

$$Q = \{x_1 x_2 x_3 \cdots x_n \text{ such that } x_i \in \mathbb{Z} \text{ and the } x_i \text{ are all distinct } \}$$

The possible start states are any random string so:

$$Q_0 = Q$$

In other words, any state can be a starting state.

The set of transitions $\delta \subseteq Q \times Q$ is:

$\{(q_1, q_2)\}$ such that q_2 is exactly the same as q_1 except two adjacent positions are swapped. In q_1 these positions were out of order (the greater one had lower index) but in q_2 the order is correct.

$\square$

(b) Prove that $NSort$ eventually terminates with S completely sorted.

Solution. Let us define three quantities:

N_{total}: Total number of out of order pairs in the string

N_{adj}: Number of out of order *adjacent* pairs in the string

N_{nonadj}: Number of out of order *non-adjacent* pairs in the string

Notice that since adjacent or non-adjacent is an exhaustive set of possibilities for a pair, for any state:

$$N_{total} = N_{adj} + N_{nonadj}$$

Also, notice that all three quantities are by definition in $\mathbb{N}$. The following lemma is very helpful:

Lemma 4.1. $N_{total} = 0 \Longleftrightarrow N_{adj} = 0$

Proof.
$\Longrightarrow$. For contradiction suppose $N_{adj} \neq 0$ but $N_{total} = 0$. Then $N_{nonadj} = -N_{adj}$. Since both N_{adj} and N_{nonadj} are nonnegative, they must both be 0. Contradiction.

$\Longleftarrow$. For contradiction, suppose $N_{adj} = 0$ but $N_{total} \neq 0$. Then there exists $i, j \leq n$ such that $i < j$ and $x_i > x_j$. Let us pick i and j to minimize their difference. In other words, out of the pairs that are inverted, let's pick the pair that is closest. Since $N_{adj} = 0$, these two entries cannot be adjacent so $j > i + 1$. So there is some entry, x_{i+1} adjacent to x_i. Since no adjacent pairs are out of order, $x_i < x_{i+1}$. Also, since x_i and x_j were the closest inverted pair, x_{i+1} and x_j cannot be inverted. Therefore, $x_{i+1} < x_j$. This means that $x_i < x_{i+1} < x_j$ which is a contradiction. ∎

First, we can use this lemma to prove **correctness**. That is, we need to show that when the algorithm terminates, the string is completely sorted. We know that our algorithm terminates when there are no adjacent inverted pairs. Applying our lemma, this means that there are no inverted pairs at all. A string with no inverted pairs is a sorted string. QED.

Second, we can use this lemma to prove **termination**. Consider the following facts:

1. N_{total} decrease with every transition. *(Note that N_{adj} does NOT decrease with every transition.)* We can show this by examining our transition. The exchange itself decreases the number of inversions by one. However, it may produce "new" adjacent inversions. But the key fact is that it can form no *new* inversions. The only thing that can happen is that a previously non-adjacent inversion can become adjacent. This is because the only items that move are the ones being exchanged, and they are being ordered correctly. As far as all the other items in the string are concerned, the two items are in the same place. Thus, the total number of inversions decreases by 1 at every transition.

2. N_{total} will eventually reach 0 in any live execution. This is because N_{total} is a nonnegative integer by definition. By the well ordering principle, and the fact that it is decreasing at every transition, it will eventually reach 0.

3. When N_{total} reaches 0, the algorithm will stop. We show this by applying our lemma. If $N_{total} = 0$ then $N_{adj} = 0$ so the algorithm terminates by definition.

Thus the algorithm will terminate. Also, at termination, $N_{total} = N_{adj} = 0$ so the string is sorted.

□

(c) Give a good upper bound on the number of steps that $NSort$ can take until it terminates.

Solution.

Two possibilities are:

Loose upper bound: Every item can be exchanged with every other item at most once. So for all n items, there are $n-1$ other items that they could be exchanged with. So $n(n-1) = O(n^2)$ is an upper bound.

Tight upper bound: N_{total} is definitely an upper bound since it decreases by 1 at every step until it reaches 0. Also, N_{total} is upper bounded by the total number of combinations of elements of the string. This is $\binom{n}{2}$ or $\frac{n(n-1)}{2}$. Notice that this is the same as the loose bound except it recognizes that every pair in the loose bound is actually being counted twice.

□

(d) For the special case where n = 4, give an example execution that takes the maximum number of steps to terminate.

Solution. The following execution takes $\frac{n(n-1)}{2} = \frac{4*3}{2} = 6$ steps:

$$
\begin{array}{rcccc}
Start \longrightarrow & 4 & 3 & 2 & 1 \\
Step\ 1 \longrightarrow & 3 & 4 & 2 & 1 \\
Step\ 2 \longrightarrow & 3 & 2 & 4 & 1 \\
Step\ 3 \longrightarrow & 3 & 2 & 1 & 4 \\
Step\ 4 \longrightarrow & 2 & 3 & 1 & 4 \\
Step\ 5 \longrightarrow & 2 & 1 & 3 & 4 \\
Step\ 6 \longrightarrow & 1 & 2 & 3 & 4
\end{array}
$$

□

Problem 5 The algorithm $Mark$ starts with a connected, undirected graph $G = (V, E)$, with all edges unmarked. Then it proceeds to mark some of the edges, by repeatedly performing the following steps:

1. **Choose** any edge $e \in E$ such that there is currently no path of marked edges between the endpoints of e

2. **Mark** edge e

Prove that when the algorithm finishes, the set of marked edges forms a spanning tree of the input graph G.

(A *spanning tree* of G is a tree whose edges are a subset of the set E of edges of G, and that contains an edge incident on every vertex of G.)

Solution.

We will need to show that the marked edges are acyclic, touch every vertex of G, and that the algorithm does indeed finish.

Marked edges are acyclic: Use the Invariant Theorem:

> **Invariant:** The marked edges form an acyclic graph.
>
> **Start condition:** Initial state. Initially all edges are unmarked, so there are no cycles consisting of marked edges.
>
> **Step condition:** Consider a step (q, q'), where we assume that there are no cycles consisting of marked edges in state q. We show that the same condition holds in state q'.
>
> In this step, the algorithm marks an edge e for which there was previously no marked path between its endpoints. This means that, after the step, there is exactly one marked path between the endpoints of e. Thus, the new edge is not part of a cycle, in the new state q'. This means that there are still no cycles in state q'.

Algorithm terminates: This is shown because once an edge is marked, it can never be unmarked. Since we have a finite set of edges, and every transition necessarily marks one edge, we will eventually run out of candidate edges and the algorithm will terminate.

Marked edges touch every vertex of G: We can show this by noticing that if (v, w) is an edge in G then there is a path of marked edges from v to w. If there wasn't, then (v, w) would become a candidate edge because its endpoints are not connected by marked edges. Then (v, w) would itself be marked and thus v and w are indeed connected by marked edges.

Now we need to show that every vertex is touched by a marked edge. For contradiction, suppose there is some $v \in G$ that is not touched by any marked edges. However, G is connected, so there must be at least one (v, w) edge in G. Since this edge exists, v and w must be connected by marked edges, so v must indeed be touched by a marked edge. Contradiction.

$\square$

Problem 6 Cache management protocol *HybridCache* is a hybrid of the two caching protocols presented in Installment 2 of the notes on computational processes. *HybridCache* maintains a memory and two caches, as before. Only now, an update to memory causes the value to be cleaned out of cache1 but not out of cache2. The intent is that cache2 is allowed to be out-of-date, but cache1 is not.

(a) Model *HybridCache* carefully as a state machine.

Solution.

Q: $\{(memory, cache1, cache2)$ of the types described in Lecture 8 handouts$\}$.

Q_0: memory is arbitrary, caches map everything to *null*.

δ: $(memory, cache1, cache2) \rightarrow (memory', cache1', cache2')$ where all steps are the same as in Lecture 8 except the memory update step:

1. $memory(a) :=$ arbitrary *value* (not null).
   $cache1(a) := null$

(b) State carefully an invariant saying that cache1 is always up-to-date.

Solution.
Can show, in all reachable states either $cache1(a) = memory(a)$ or $cache1(a) = null$

In other words: If $cache1(a) \neq null$ then $cache1(a) = memory(a)$.

Note: No such guarantees are provided for cache2.

(c) Prove your invariant carefully using the Invariant Theorem.

Solution.

Start condition: Everything in cache1 is null so the invariant is trivially true.

Transition condition: We show all possible transitions:

1. **Memory Update:** After this step, the only changed value is in cache1 and is changed to null. In this case, the Invariant holds trivially (they only need to match if cache1 holds a value other than null).
2. **Cache1 Update:** The only changed value is in cache1 and is changed to match the one in memory. Invariant holds.
3. **Cache2 Update:** Only cache2 changes. Since the Invariant makes no claims about cache2, the Invariant still holds.
4. **Cache1 Drop Value:** The only changed value is in cache1 and is changed to null. In this case, the Invariant holds trivially.

5. Cache2 Drop Value: Same as 3.

(d) Now modify $HybridCache$ to a new $WrongHybridCache$ algorithm, which is just like $HybridCache$ but includes one additional type of step, which allows anything to be copied from cache2 to (the same address in) cache 1. Model this new algorithm as a state machine.

Solution.

Same as in a. except for one additional transition:

6. $cache1(a) := cache2(a)$

(e) Give an execution of $WrongHybridCache$ that demonstrates that the invariant is violated.

Solution.

memory_write(1)
$memory(a) := 1$
$cache1(a) := null$

cache1_update()
$cache1(a) := 1$

cache2_update()
$cache2(a) := 1$

memory_write(2)
$memory(a) := 2$
$cache1(a) := null$

cache2_to_cache1()
$cache1(a) := 1$

Now cache1 holds a different value than memory in entry a.

(f) Identify precisely where your proof of the invariant for $HybridCache$ would break down if you applied it to $WrongHybridCache$.

Solution.

It would break down in trying to prove the transition holds through transition 6. Here it is not the case that the value in cache1 is always changed to either match memory or be null. Cache1 can take on any value from cache2, and we have no guarantees that the value in cache2 is up to date.

□

Problem 7 It turns out that the stable marriage algorithm described in lecture (Installment 2 of notes) is very unfair, in the sense that, in any (live) execution of the algorithm:

- The boys end up paired up with the most preferred girl that they possibly could get.
- The girls end up paired up with the least preferred boy that they possibly could get.

(So, being aggressive pays.)

More precisely, define PP ("possible pairs") to be the set of all (boy,girl) pairs that occur in any stable pairings. Then, in the final state of the algorithm:

- If a boy b likes a girl g more than the partner he has received, then $(b, g) \notin PP$, that is, g isn't even possible for him.
- If a girl g likes a boy b less than the partner she has received, then $(b, g) \notin PP$.

Prove these results.

Solution.

(a) You can prove as an invariant, the claim that: "If b has crossed g off his list, then $(b, g) \notin PP$." This statement immediately implies the claim in (a).

The key step is where g rejects b, and he crosses her off his list. She rejects him in favor of some other boy b'.

Now, suppose for the sake of contradiction that P is a stable pairing containing (b, g). We can show that (b', g) form an unstable pair in P: We know that g prefers b' to b, because in this step of the algorithm, she rejects b in favor of b'. We have to argue that, also, b' prefers g to whomever he is matched up with in P—call her g'.

This is a tricky argument. Can argue it by contradiction, using the fact that the invariant is true before the step. Suppose for the sake of contradiction that b' prefers g' to g. In the pre-state, b' is pursing g, so he must have crossed off his higher preference g'. But by the invariant in the pre-state, this means that $(b', g') \notin PP$, that is, (b', g') do not form a possible pair. That's a contradiction, because we've assumed that they are paired in the stable pairing PP.

(b) You can prove the invariant:

"If g likes b less than her most-preferred boy that is currently pursuing her, then $(b, g) \notin PP$." This statement implies the claim in (b).

This can be proved without using an inductive argument (or the Invariant Theorem), but simply by contradiction using the invariant used for part (a).

Suppose for the sake of contradiction, that there is some reachable state in which g likes b less than her most-preferred boy that is currently pursuing her, call him b', but $(b, g) \in PP$. Fix P, some stable pairing having $(b, g) \in P$. We can show that (b', g) form an unstable pair in P: We know that g prefers b' to b, by the assumption above. We have to argue that also b' prefers g to whomever he is matched with in P, call her g'.

Suppose he doesn't (this is going to be a contradiction within a contradiction!), that is b' prefers g' to g. That is, b' prefers g' to the girl he is currently pursuing. This means that b' has crossed g' off his list in the current state. By the previous invariant, proved for part (a), it follows that $(b', g') \notin PP$. So surely we can't have $(b', g') \in P$. This is a contradiction, which means that b' does prefer g to g'. That's what we needed to show above.

So, this all says that (b', g) form an unstable pair in P. That's a contradiction to the assumption that P is a stable matching.

Problem 8 Consider an "abstract object" that maintains a dynamic polynomial *poly*. This polynomial has variable x, and integer coefficients. Initially, the polynomial is just the 0 polynomial (no terms). The polynomial supports operations:

- $\text{add} - \text{term}(c, e)$, which is supposed to add the term cx^e to the maintained polynomial.
- $\text{coeff}(e)$, which asks for the coefficient of the x^e term.

Solution.

Solution to this problem will be given out next week. The material covered in this problem will not be covered on the quiz.
