CS170–Fall 2014 — Solutions to Homework 1

Howard Chen, SID 23665755, `cs170-ed`, Collaborators: Kevin Wu

September 5, 2014

1. Getting started

(a) I understand the course policies

(b) No this is not allowed. Everyone has to write up their own solution.

2. Proof of correctness

So the invariant at the beginning of the loop is

$$y_i = \sum_{k=0}^{i} a_{k+n-i}c^k$$

Base Case: When i = 0 in our algorithm the value is a_n. If we plug in i = 0 in our invariant we also get a_n so our base case is true.

Inductive Hypothesis: Our invariant holds true at y_i

Inductive Step: in our algorithm at the begining of loop iteration $i + 1$, our value

$y_{i+i} = y_i * c + a_{n-i}$

$$= \sum_{k=0}^{i} a_{k+n-i}c^k * c + a_{n-i}$$

$$= \sum_{k=0}^{i} a_{k+n-i}c^{k+1} + a_{n-i}$$

$$= \sum_{k=0}^{i} a_{k+n-i}c^{k+1} + a_{n-i} * x^0$$

If we expand the summation out we can see that it is our invariant but all the exponents are increased by 1, so adding the second term of $a_{n-i} * x^0$ will be the same as

$$y_{i+1} = \sum_{k=0}^{i+1} a_{k+n-i}c^k$$

Therefore the invariant is correct.

Since the invariant is true we know the loop gets the correct summation of all the polynomial's terms by the end of the loop since it iterates over all the terms it will be correct. Basically i will = n by the finishing of the loop thus my invariant equals

$$y = \sum_{k=0}^{n} a_k c^k$$

3. Prove this algorithm correct

(a) c can never go negative because first c starts out at 0. When it goes to the first iteration of the loop, c will automatically be set to 1 in the following statement. Then c can either be increased or decreased by one. This means c is only ever at risk of turning negative when it is at 0. However, like said earlier, once it hits zero, the logic will force it to increment to 1, thus having a negative c is impossible.

(b) Basecase
At The beginning of the algorithm it holds true because after we run the loop at i = 0 our list is just A[0], which only has one element. Thus $U_1 = [x]$ and $P_1 = []$ which holds true to the invariant.

Inductive hypothesis
That we can partition the array up to index $i-1$ and split it into a group that contains atleast c instances of value v and a group that can be paired off so that the two elements in each pair differ.

Inductive Step

Case c = 0
If c = 0 then that means in the previous step we had a no match. This means that U is currently empty because c = 0, and we can add the current A[i] at to U at the end of the loop.

Case c ¿ 0 and v = A[i]
We know is a match so we can append A[i] to the list U.

Case c ¿ 0 and v != A[i]
We know that there is a mismatch so we can remove the element an element from U and add that element paired with A[i] to P.

(c) This algorithm is correct because as we go along the list the elements are always added to either P or U, and in the case we add to P, we pull an element from U in order to finish the pair. This means that within U and P are elements from our list and thus our output must exist in our list. Since the definition of majority is greater than 50% that means every element paired element in P will neutralize the element it's paired up with to find a majority. If an element is truly a majority of the list then in the end its elements will be paired with other elements that aren't equivalent in P and have some remainder in U, since our invariant shows us that our list will be divided into those two if we go through the whole list, which is what the algorithm does. That means its unpaired elements (which occurs since it is a majority) will exist in U, which by the definition of U is the value v, which is what the algorithm returns, making it correct. If there isn't a majority, the algorithm can output anything it wants, so it can output anything and still be correct.

4. Check my Proof

(a) The proof is false because the good edges don't actually increase, but the bad edges decrease.