

CS215 ASSIGNMENT 3 SOLUTIONS

(was due Thursday, Nov. 4, 8AM)

Problem 1. Which of the following instances of the PCP have matches? Describe a match if it exists, or prove that it doesn't.

$$(a) \begin{array}{l} bba \quad bb \quad ab \\ b \quad , \quad abb \quad , \quad bba \end{array}$$

$$(b) \begin{array}{l} abb \quad a \quad abb \quad ba \\ ab \quad , \quad baa \quad , \quad b \quad , \quad babba \end{array}$$

$$(c) \begin{array}{l} abb \quad bba \quad abab \quad bba \quad ab \quad babb \quad bbaba \quad bbabb \\ a \quad , \quad bb \quad , \quad ab \quad , \quad ba \quad , \quad bba \quad , \quad bba \quad , \quad aba \quad , \quad abb \end{array}$$

Solutions. (a) This set will have a match if and only if the set $\begin{array}{l} abb \quad bb \quad ba \\ b \quad , \quad bba \quad , \quad abb \end{array}$ has a match. (This is because this set is obtained by reversing all the strings in the first set.)

Any match for this set would have to start with $\begin{array}{l} bb \\ bba \end{array}$, because otherwise the first pair of characters (top and bottom) would not match. After that, the next pair would have to be $\begin{array}{l} abb \\ b \end{array}$, because the a on the bottom of $\begin{array}{l} bb \\ bba \end{array}$ would have to be matched by the next character added to the top. So, so far we know any match would have to start like this: $\begin{array}{l} bbabb \dots \\ bbab \dots \end{array}$.

After that, to match the last b on the top, we have to use the pair $\begin{array}{l} bb \\ bba \end{array}$ or the pair $\begin{array}{l} abb \\ b \end{array}$. Adding the first one cannot lead to a match because it would give $\begin{array}{l} bbabbbb \dots \\ bbabbbb \dots \end{array}$ in which the last character does not match. Thus, we would have to add the second one, giving $\begin{array}{l} bbabbabb \dots \\ bbabb \dots \end{array}$. To match the abb on the bottom, the only pair to add would be $\begin{array}{l} ba \\ abb \end{array}$, which would give $\begin{array}{l} bbabbabbba \dots \\ bbabbabb \dots \end{array}$.

After that, the only pair to add would be $\begin{array}{l} abb \\ b \end{array}$, which would give $\begin{array}{l} bbabbabbbaabb \dots \\ bbabbabb \dots \end{array}$.

At that point, there is no pair that could be added. Thus, there is no way to make a match.

(b) For convenience we refer to the dominoes by number:

$$1 = \begin{array}{l} abb \\ ab \end{array}, \quad 2 = \begin{array}{l} a \\ baa \end{array}, \quad 3 = \begin{array}{l} abb \\ b \end{array}, \quad 4 = \begin{array}{l} ba \\ babba \end{array}.$$

First, note that neither domino 2 nor 4 can end a match. (The reason is that if, going from the back, the unmatched part of the bottom string ends with ab there is no way to continue. If the last domino is 4, you get stuck right away. If the last domino is 2, then the next-to-last domino has to be 4, so the end of the match would have to look like $\dots \begin{array}{l} \dots baa \\ \dots babba \end{array}$. The domino before that has to be 4 again, giving $\dots \begin{array}{l} \dots babaa \\ \dots babbababba \end{array}$. But now there is no domino to put before that.)

Suppose there is a match. Consider a match that uses the smallest possible number of dominoes. Consider placing the dominos in this match from left to right one by one. The first domino placed has to be 1. After that, consider the first time that the dominoes placed so far make the bottom string as long or longer than the top string. (Perhaps this doesn't happen until the last domino is placed, but it definitely happens at some point.) When this happens, the domino placed must be either 2 or 4 (because these are the only ones with longer bottom strings).

We claim first that this domino cannot be 4 ($\begin{array}{l} ba \\ babba \end{array}$). (Suppose it was 4. The ba in the top of 4 does not match the first ba in the bottom of 4, because then the dominoes placed before this domino would give a shorter match, which would contradict our assumption that we are considering

the shortest match. And the ba in the top cannot match the second ba in the bottom, because then we would have a match ending in 4, which we know cannot happen.)

So the domino would have to be $2 \begin{pmatrix} a \\ baa \end{pmatrix}$.

The a in the top of 2 cannot match the last a in the bottom of 2, because then we'd have a match, but a match cannot end with domino 2.

So the a in the top of 2 matches the middle a in the bottom of 2. Then the previously placed dominoes must form the pair of strings $\begin{matrix} xb \\ x \end{matrix}$ for some string x . The only domino that can be placed next is domino 3, $\begin{pmatrix} abb \\ b \end{pmatrix}$. After the two dominoes 2 and 3 are added, and the rest, we get $\begin{matrix} xbaabb \cdots \\ xbaab \cdots \end{matrix}$, so the remainder of the match has to form the strings $\begin{matrix} y \\ by \end{matrix}$ for some string y . But then the strings $\begin{matrix} xby \\ xby \end{matrix}$ also form a match, and this match is obtainable by deleting the dominoes 2 and 3 just placed. But this contradicts our assumption that we are considering the shortest match.

(c) First, we can eliminate the second and third dominoes because they have more a's in top than in bottom, while all other have the same number. (If the second or third domino were ever used, the number of a's on the top of the match would have to exceed the number of a's on the bottom, which cannot happen.)

So if there is a match, there is one using the following dominoes:

$$\begin{matrix} abb & bba & ab & babb & bbaba & bbabb \\ a & ba & bba & bba & aba & abb \end{matrix}$$

The only way to start is with the first domino, and now the top has an unmatched string bb . The only way to continue is to add the third domino, and then the fourth. But now the top has an unmatched string bb again, so again we would have to add the third and fourth dominoes...

Problem 2. A set of vertices D in an undirected graph $G = (V, E)$ is called a *dominating set* if every vertex either belongs to D or has a neighbor in D . The Dominating Set Problem is:

DOMSET:

Instance: graph G , integer K ;

Query: does G have a dominating set of size K ?

Below I present two polynomial-time reductions. Are these reductions correct? For each reduction, if the reduction is correct, present the correctness proof. If the reduction is incorrect, give a complete explanation of what the error is.

Reduction A. This is a reduction from 3SAT. (See Sipser for definition of 3SAT.) Let α be a given boolean expression in 3cnf. Let G be the graph that consists of two vertices and no edges. If α is satisfiable, we map α into (G, K) , where $K = 2$. If α is not satisfiable, we map it into (G, K) , where $K = 1$.

Reduction B. This reduction is from VERTEX-COVER. (See Sipser for definition of VERTEX-COVER.) The reduction is very simple: given any instance (G, K) of VERTEX-COVER, we map it into the same pair (G, K) as the instance of DOMSET.

Solution. A function f gives a polynomial-time reduction from A to B if and only if

- (1) f is computable in polynomial time
- (2) $\forall w \in \Sigma^*, w \in A \iff f(w) \in B$.

The function for part A is

$$f(\alpha) = \begin{cases} (G, 2) & \text{if } \alpha \text{ is satisfiable} \\ (G, 1) & \text{if } \alpha \text{ is not satisfiable.} \end{cases}$$

This reduction satisfies the second property ($w \in A \iff f(w) \in B$) but the description fails to say how to compute the function f in polynomial time. In particular, given α , it is not clear how to tell whether α is satisfiable in polynomial time.

Since in fact no one knows whether it is possible to determine this in polynomial time, we can certainly say that the description of the reduction is inadequate and that the idea behind it is insufficient. (Technically, we don't know whether the reduction described is a polynomial-time reduction or not, although in spirit the reduction is certainly correct.)

The proposed reduction for part B fails the second necessary property. For example, take G to be the graph with two vertices and no edges, and take $K = 1$. Then the graph has a vertex cover of size K , so $(G, K) \in \text{VERTEX-COVER}$, but G has no dominating set of size 1, so $(G, K) \notin \text{DOM-SET}$.

Problem 3. Consider the following two versions of SUBSET-SUM: (See Sipser for definition of SUBSET-SUM.)

SS1: this is SUBSET-SUM with the additional assumption that $x_{i+1} \geq 2x_i$ for $i = 1, \dots, k-1$.

SS2: this is SUBSET-SUM with the additional assumption that all x_i have only two possible values (that is, there are a, b such that each x_i is either a or b)

For each of SS1 and SS2, give a polynomial-time algorithm (the faster the better) that works on instances that meet the stated assumptions.

Solutions. The problem as described is flawed. To solve SS1 in polynomial time, we need the additional assumption that the x_i 's are non-negative (otherwise the problem is as hard as the original problem, which is an interesting exercise to show). So we assume that the numbers are non-negative.

Consider the following algorithm for SS1:

```
SOLVE-SS1( $T, X_1, X_2, \dots, X_k$ ):
1. if  $T < 0$  return 'reject'
2. if  $T = 0$  return 'accept'
3. if  $K = 0$  return 'reject'
4. if  $X_k > T$  return SOLVE-SS1( $T, X_1, X_2, \dots, X_{k-1}$ )
5. return SOLVE-SS1( $T - X_k, X_1, X_2, \dots, X_{k-1}$ )
```

The algorithm runs in polynomial time because it makes at most k recursive calls.

To finish we prove that the algorithm correctly decides SS1, assuming that all the X_i 's are non-negative and $X_{i+1} \geq 2x_i$.

If $T < 0$, then clearly no subset of the X_i 's sums to T (since they are all non-negative), so the algorithm is correct to return 'reject'.

If $T = 0$, then the empty subset of the X_i 's sums to T so the algorithm is correct to return 'accept'.

Otherwise ($T > 0$) if $K = 0$ then there are no X_i 's, so the only subset possible is the empty set, which sums to zero, which is not T , to the algorithm is correct to return 'reject'.

Otherwise $T > 0$ and $K > 0$. Consider this case.

If $X_k > T$, then clearly there is a subset S that sums to T if and only if there is a subset S of $\{X_1, \dots, X_{k-1}\}$ that sums to T . By induction on k , we can assume that the recursive call in step 4 correctly determines whether this is true. So in this case the algorithm returns the correct answer.

If $X_k \leq T$, then we claim any subset S that sums to T must contain X_k . (To see why, note that the sum of the elements X_1, X_2, \dots, X_{k-1} is at most

$$X_k/2^{k-1} + X_k/2^{k-2} + \dots + X_k/4 + X_k/2 < X_k(1/2 + 1/4 + 1/8 + \dots) = X_k \leq T.$$

So even if we take all of the remaining elements in the subset, the sum is not as large as T .)

Thus, if $X_k > T$, then there is a subset S that sums to T iff there is a subset S containing X_k that sums to T . And this is true iff there is a subset S' of $\{X_1, \dots, X_{k-1}\}$ that sums to $T - X_k$. By induction, we can assume that the recursive call in step 4 correctly determines whether this is true. So in this case the algorithm returns the correct answer.

Solution for SS2. Consider the following algorithm for SS2:

SOLVE-SS2(T, X_1, X_2, \dots, X_k):

1. Find a and b such that each X_i is either a or b .
2. Let I be the number of a 's and J be the number of b 's.
3. For each $i \in \{0, 1, 2, \dots, I\}$ and each $j \in \{0, 1, 2, \dots, J\}$ do
4. If $i \times a + j \times b = T$, return 'accept'
5. return 'reject'

The algorithm runs in polynomial time because the loop tries $O(k^2)$ pairs of values for i and j .

The algorithm is correct because of the following observation:

There is a subset summing to T if and only if there are $i \in \{0, 1, \dots, I\}$ and $j \in \{0, 1, \dots, J\}$ such that $i \times a + j \times b = T$.

Since the algorithm checks the condition for all possible such i and j , the observation implies that the algorithm is correct.

At the risk of proving the obvious, we will verify carefully that the observation is true.

First, suppose there is a subset S summing to T . Let i be the number of a 's in S and let j be the number of b 's in S . Then: (1) Clearly $0 \leq i \leq I$ and $0 \leq j \leq J$. (2) Since only a 's and b 's are in S , the sum T of the numbers in S , equals $i \times a + j \times b$. Thus, if there is a subset S summing to T , then there are $i \in \{0, 1, \dots, I\}$ and $j \in \{0, 1, \dots, J\}$ such that $i \times a + j \times b = T$.

Conversely, suppose there are $i \in \{0, 1, \dots, I\}$ and $j \in \{0, 1, \dots, J\}$ such that $i \times a + j \times b = T$. Make a subset S by taking i a 's and j b 's from $\{X_1, X_2, \dots, X_k\}$. Clearly we can do this, and this subset S clearly sums to T . Thus, if there are $i \in \{0, 1, \dots, I\}$ and $j \in \{0, 1, \dots, J\}$ such that $i \times a + j \times b = T$, then there is a subset S summing to T .