CS14 Winter 2016 Lab 1
Making Cash Payments with Martian Square Money

January 7, 2016

DUE: Submit by iLearn by 11:59pm Sunday, January 17, 2016
LATE SUBMISSION (50% penalty): 11:59pm Monday, January 18, 2016

1 Introduction

In order to prevent the merchant class from becoming too powerful, the ancient Martian rulers devised a system of currency to make it difficult for the merchants to carry out business in which the coin denominations were (literally) *all perfect squares*. More specifically, they minted coins valued at 1μ, 4μ, 9μ, 16μ, 25μ, 36μ, and so on. Furthermore, they decreed that *all money transactions are* **illegal** *unless they are expressed as a coin vector using the* **smallest possible number of coins.** In this assignment, you will be writing a currency converter between Earth notation and Martian notation.

2 Coin Vectors

A Martian coin vector starts with the keyword MU, followed by a series of non-negative integers separated by commas. The first integer gives the number of 1μ coins, the second integer gives the number of 4μ coins, and so on. Trailing zeros are dropped to minimize the length of the vector, so the rightmost integer is always positive except for the zero amount (written as MU 0). However, leading zeros are *always required* since the individual coin denominations are only identified by their positions in the vector.

For comparison purposes, let us define *Earth notation* to consist of the keyword EA followed by a single integer, indicating the total amount for the transaction. Thus, the amount EA 12 in Earth notation could be written as five different vectors in Martian notation, namely:

(a) MU 3,0,1 = three 1μ and one 9μ coins
(b) MU 0,3 = three 4μ coins
(c) MU 4,2 = four 1μ and two 4μ coins
(d) MU 8,1 = eight 1μ and one 4μ coins
(e) MU 12 = twelve 1μ coins

However, only vector (b) is legal because it uses a total of 3 coins versus 4 coins for vector (a), 6 coins for vector (c), 9 coins for vector (d), and 12 coins for vector (e).

3 Conversion into Martian Notation

You may have studied the algorithm for making change using one of Earth's decimal currencies, where larger denominations are almost always whole multiples of smaller ones.[1] In this case, the rule for making a transaction amount using the fewest total number of coins is to try the denominations in order from largest to smallest, and include as many as possible of each denomination before moving on to the smaller ones. For example \$0.97 would be formed using three \$0.25, two \$0.10 and two \$0.01.

[1] Except for cases like the 25 cent coin not being divisible by the 10 cent coin, and the 50 dollar bill not being divisible by the 20 dollar bill.

Unfortunately, the standard Earth algorithm for making change does *not* always give the best result for Martian Square currency, because the different coin denominations fit together rather badly. Indeed, the previous example of converting EA 12 already provides a counter-example to its optimality, since including the largest-possible 9μ coin makes things worse.

One way that is always guaranteed to give you the optimal Martian coin vector is the following "brute-force" approach, which simply tries every possibility:

1. Find the largest denomination coin that fits within the transaction amount, say the nth coin with value $n^2\mu$.

2. Choose each possible number of the nth coin in turn, and subtract their combined value from the transaction amount.

3. For each choice of the number of the nth coin, choose each possible number of the $(n-1)$st coin in turn, and subtract their combined value from the remaining transaction amount after step 2.

4. Continue the pattern shown in step 3 for choosing each possible number of the $(n-2)$nd coin for each combination of the nth and $(n-1)$st coins, then choosing each possible number of the $(n-3)$rd, for each combination of the nth through $(n-2)$nd coins, and so on down to the 2nd (4μ) coin.

5. Use the 1μ coin to make up all of the remaining transaction amount. Count up the total number of coins used for this coin vector and save a copy if it is the best one you have found so far.

Notice that the sequence of coin vectors shown above for converting EA 12 into Martian Square currency follows this algorithm, if you assume that the choices for each coin are tested in *decreasing order* from the maximum to zero as we proceed down the list.

Note also that the brute force algorithm can be programmed using $(n-1)$ nested loops, and thus executing the code will become very time consuming as the transaction amount increases. One of the primary goals of CS14 is the subject of evaluating the *quality* of algorithms — as opposed to merely judging their *correctness*. Not only will we learn how to estimate the *complexity* of algorithm — i.e., how much time and memory they require as a function of the problem size — but we will also study "good" algorithms for certain types of problem.

4 Algorithm Improvements

We will talk more about the Martian Square currency conversion problem later in the term. For now, however, here are a few improvements to the above "brute force" algorithm you can add to your code to make it better without any risk of getting the wrong answer.

- **Give up early on a bad choice.** As written above, the algorithm does not calculate the total number of coins in the coin vector until step 5, where the number of coins of every denomination has been determined. But suppose you already found one solution that uses a total of K coins and right now you are only part way through the creation of another coin vector and it already contains more than K coins. Clearly the current coin vector will be discarded at the end of step 5, so why bother to finish its construction?

- **Use biggest-first search order.** The "brute force"algorithm merely requires us to try all possible ways of making up the transaction value out of the available set of coin denominations. Thus, we could organize the code to choose the *smallest* number of coins of each denomination first at each step, rather than the *largest* number. If we did that, then the earliest coin vectors we find would avoid using large-denomination coins (which are chosen first), and be forced to replace them by large numbers of smaller-denomination coins to make up the total transaction amount. Even though these coin vectors are very unlikely to provide the optimal solution, we can't apply the previous improvement to up on them early unless we test them *after* finding a better solution.

- **Zero remainder is always the best choice.** Clearly, if the transaction amount is a perfect square such as EA n^2, then the optimal Martian coin vector will contain a single $n^2\mu$ coin, since we cannot represent a non-zero amount using zero coins. More generally, suppose at some step of the "brute force" algorithm, the *remaining* transaction amount is *evenly divisible* by the current coin denomination. In that case, the best coin vector we can obtain by continuing this example is to make up the remainder of the transaction immediately using only the current coin denomination. Including any smaller-denomination coins will only make things worse.

  To see why this is true, consider the problem of converting EA 72 to a Martian coin vector, and assume that we have just chosen to use zero 64μ coins at step 2 and chosen to use zero 49μ coins at step 3. In this case, we finish step 3 with a remaining transaction amount of EA 72, which must be made up entirely from some combination of 36μ, 25μ, 16μ, 9μ, 4μ, and 1μ coins during step 4. At this point, we have the option to choose two, one or zero 36μ coins — and leave us with a remaining transaction amount of EA 0, EA 36, or EA 72, respectively, which must be made up by some combination of even-lower denomination coins. If we choose two 36μ coins then we know that the final coin vector will consist of exactly two coins because the remaining transaction balance is now zero. However, if we choose fewer than two 36μ coins, then the total number of coins in the vector must increase because the value of each replacement for a 36μ coin must have a smaller denomination, so the remaining transaction balance will be greater than zero unless we add more coins to the vector.

5 Program Specifications

Write a program that performs currency conversions *in both directions* between Earth notation and Martian Square vectors. Your program must accept command line inputs consisting of an optional *flag* "-v" followed by one *input file*, and produce one similary-named *output file* with the original filename extension (if any) changed to "`.out`".

For example, if the input file is `ass1.txt`, then the output file is `ass1.out`. The input is a plain ASCII text file, consisting of a variable number of rows until *end-of-file*. Each row consists of a single transaction, represented in either Earth or Martian Square notation.

- A transaction in *Earth notation* consists of the letters EA followed by white space, followed a single integer representing the total amount.

- A transaction in *Martian Square notation* consists of the letters MU followed by white space, followed by a sequence of integers separated by commas (and, optionally, white space). The ith the integer in the sequence represents the number of coins of denominations $i^2\mu$.

For each input transaction, your program must identify its transaction type, and then convert the transaction to the opposite type.

If the optional "-v" flag is *not* present, then the program should write one line in the output file for each line from the input file. This output line should be a copy of the original input file, followed by the string " = ", and finally by the result of converting the entry to the other format. For example, if the input file consists of the following two lines:

```
EA 13
MU 0,1,0,1
```

then the output file should consist of the following two lines:

```
EA 13 = MU 0,1,1
MU 0,1,0,1 = EA 20
```

If the optional "-v" flag is present, then your program should generate *verbose* output that shows all the intermediate results (legal or not) that it generated during each the conversion from Earth notation to Martian Square notation. For example, if the input file contained a single line with `EA 13`, then a naïve program that ignores all of the algorithm improvements from section 4 would write the following sequence of lines into the output file:

```
EA 13 could be represented as:
MU 0,1,1
MU 4,0,1
MU 1,3
MU 5,2
MU 9,1
MU 13
A legal representation for this amount is:
MU 0,1,1
```

If your program includes some of the algorithm improvements from section 4, then it might avoid generating some of the possible the coin vectors. In this case, it should indicate that these "short-cuts" were employed by replacing the (unknown) number of unevaluated low-denomination coins by the star "*" character. For example, if we applied the 'Give up early on a bad choice' improvement to the conversion of EA 13 to Martian Square notation, then the output would become:

```
EA 13 could be represented as:
MU 0,1,1
MU 4,0,1
MU *,3
MU *,2
MU 9,1
MU 13
A legal representation for this amount is:
MU 0,1,1
```

These star characters appear because the algorithm knows the optimal solution contains at most two coins after generating the first vector. Thus, during the evaluation of the 3rd and 4th vectors, the algoritm would recognizes that each vector already contains at least two coins whose denomination is greater than 1μ, before considering the number of 1μ coins for these cases.