Scribed by Chaitanya Garg

October 7, 2013 – Part 2

Presentation on Nancy Leveson by Gabrielle

Here we take a quick break looking at Professor Shasha using Unix like a champ.

Presentation on Adrian Stoica by Alexandra

Presentation on Jake Loveless by Noah

Python: please see the code emailed by Professor Shasha named coins1.py to understand this section, it will also be edited and marked up by me. Edited/marked up file is named coins.py

Coin denominations
For every possible cost, evaluate the minimum number of coins to get to the point. In the US currency system, that's quite easy as each value is at least 2 times greater than the other, so just take off the biggest coin possible and repeat. But in other currency systems (that are generally made up because they're impractical), a different algorithm must be used.

Strategy:
Initialized Array:

Location	0	1	2	3	4	5	6	7	8
Amount of coins required	100	100	100	100	100	100	100	100	100
What coins are required									

The 100 represents the amount of coins required. Of course this is totally false, it's just a high value to start to compare others to in order to lessen this value down.
The "what coins are required" tells you what actual coins you should use (eg a 1 cent coin, a 9 cent coin, whatever).

i represents the current location in the array. Current location also represents the amount of money you want. So location 25 represents 25 cents total. If i is inside the denomination array, you know that you just need one coin in there, the one in the denomination array. Example if i is 25 and we're using US currency, you know you just need one coin, a 25-cent coin aka quarter.

If i not in denomination array:
You split the searches. Let's say you want to get to 33 cents. You can go from 0 to 20 and 20 to 13. And you brute force this out. So 0 to 1 and 0 to 32, then 0 to 2 and 0 to 31, then 0 to 3 and 3 to 30 so on so forth. And you pick the one that requires the smallest amount of coins, comparing it to the amount that was previously required (initialized to the high value of 100).

Example with 7 cents in US coin denominations. Assume we've filled out 1-6 so let's try with i=7.

Location	0	1	2	3	4	5	6	7
Amount of Coins	0	1	2	3	4	1	2	100

Going through the code:

k = 1. i-k = 6. Add coinnum[k] and coinnum[i-k] or 1+2, and get 3. Is 3 less than 100? Yes, and store 3 into coinnum[i].
k = 2. i-k = 5. Add coinnum[k] and coinnum[i-k] or 2+1, and get 3. Is 3 less than 3? No. Move on.
k = 3. i-k = 4. 3+4=7. 7 < 3? No. Move on.
k = 4. i-k = 3. 4+3=7. 7 < 3? No. Move on.

At this point k is no longer less than 1+i/2, the loop condition (if you continued you'd just repeat your calculations). So the lowest value you found of coins required is 3. Fill that in at location 7. So you need a minimum of 3 coins to get to 7 cents.

This technique is called dynamic programming where you recursively go through the data and end up storing previously calculated values in an area rather than recalculating them recursively (the fibslow from last program is very slow as it uses recursion but not dynamic programming).

Assignment: you get three denominations, or three types of coins. The assignment is to calculate the number of coins required from 1-99 and minimize that value (as in add up the amount of coins required to get to 1 cent, 2 cents, 3 cents, etc. up to 99 cents, and minimize that value).

General method: search through all different combos, but you know one of them has to be 1 cent coin. So 2 other coins to test through (eg a 6 cent coin and a 34 cent coin). And run this same algorithm which is given by Professor Shasha to get amount of coins required to get to each cent value.

Homework: make XOR circuit, read next 2 chapters of Natural Computing, and Python assignment listed under "Assignment"