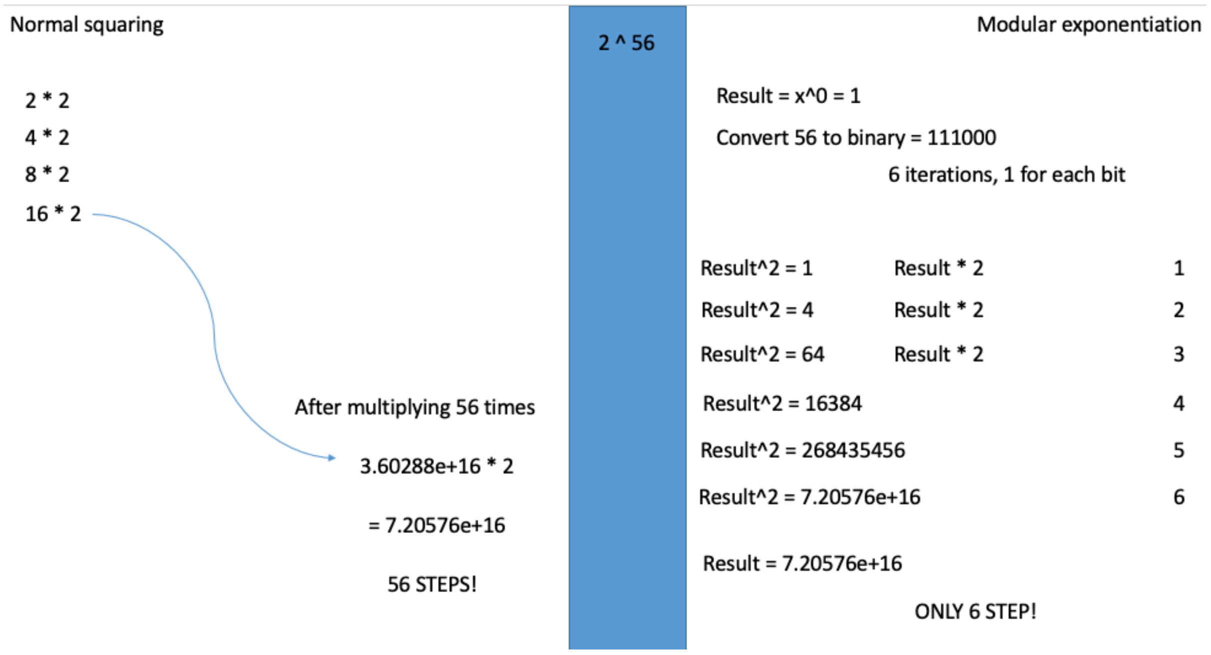
Algorithms essay

Modular exponentiation by repeated squaring

This algorithm is a fast way to compute large exponent results with very few steps, reducing the number of calculations needed to reach the final answer.
The standard way to calculate exponents would be to continuously multiply the value by itself, x number of times.

The algorithm works firstly by converting the exponent into binary, e.g. x^9; 9 = 1001. The binary number is then looped over (4 times in the case of 9), each time squaring the result and multiplying the value by x (only if the binary value being looped on is 1, not 0) e.g. see below.



Normal squaring

2 * 2
4 * 2
8 * 2
16 * 2

After multiplying 56 times

3.60288e+16 * 2

= 7.20576e+16

56 STEPS!

2 ^ 56

Modular exponentiation

Result = x^0 = 1

Convert 56 to binary = 111000

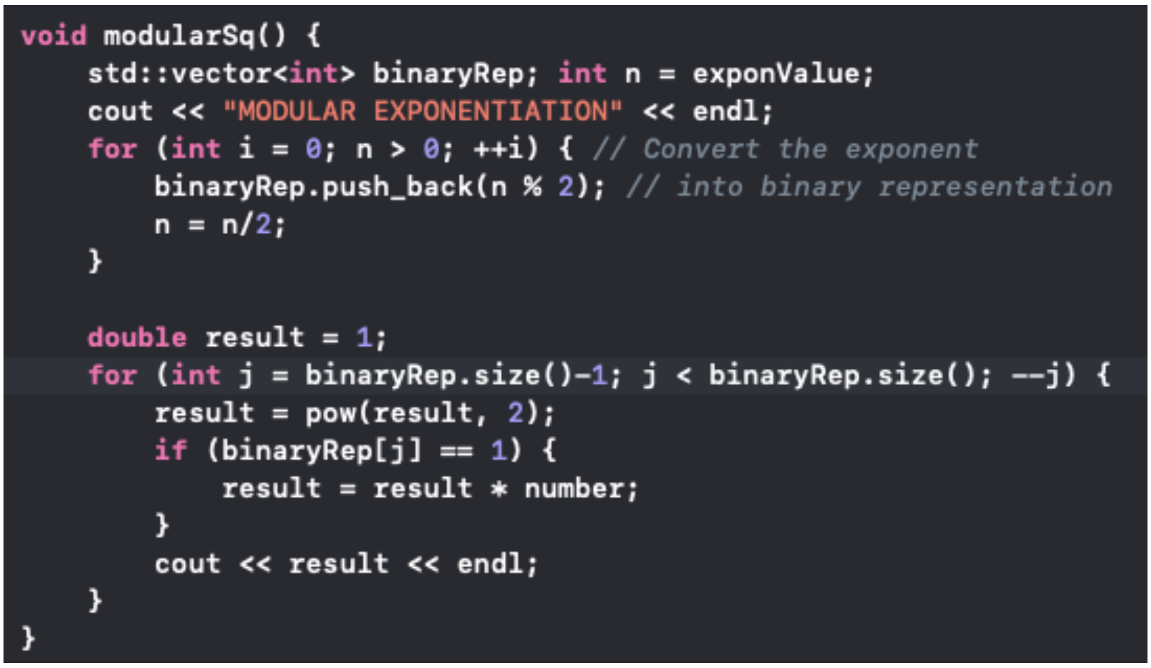
6 iterations, 1 for each bit

Result^2 = 1	Result * 2	1
Result^2 = 4	Result * 2	2
Result^2 = 64	Result * 2	3
Result^2 = 16384		4
Result^2 = 268435456		5
Result^2 = 7.20576e+16		6

Result = 7.20576e+16

ONLY 6 STEP!

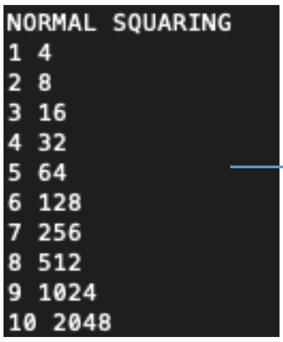
This means that the number of steps is cut massively however, the number of calculations per iteration is 2 (or 1 in case of binary number 0).



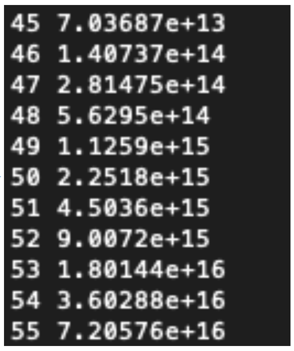
```
void modularSq() {
    std::vector<int> binaryRep; int n = exponValue;
    cout << "MODULAR EXPONENTIATION" << endl;
    for (int i = 0; n > 0; ++i) { // Convert the exponent
        binaryRep.push_back(n % 2); // into binary representation
        n = n/2;
    }

    double result = 1;
    for (int j = binaryRep.size()-1; j < binaryRep.size(); --j) {
        result = pow(result, 2);
        if (binaryRep[j] == 1) {
            result = result * number;
        }
        cout << result << endl;
    }
}
```

Algorithms essay

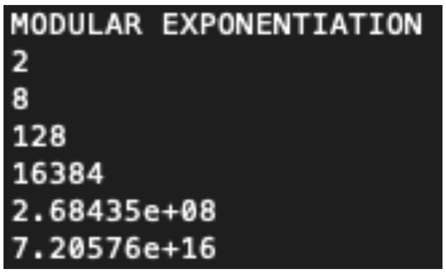
Above is my implementation of the modular exponentiation algorithm, along with the output comparison of each iteration with the standard calculation. (55 steps to 6)



```
NORMAL SQUARING
1 4
2 8
3 16
4 32
5 64
6 128
7 256
8 512
9 1024
10 2048
```



```
45 7.03687e+13
46 1.40737e+14
47 2.81475e+14
48 5.6295e+14
49 1.1259e+15
50 2.2518e+15
51 4.5036e+15
52 9.0072e+15
53 1.80144e+16
54 3.60288e+16
55 7.20576e+16
```



```
MODULAR EXPONENTIATION
2
8
128
16384
2.68435e+08
7.20576e+16
```

The modular exponentiation algorithm can be used in cryptography and is used to calculate modulus values e.g. in the range of 0 – 12345. This can then be applied to ‘code’ messages and data to stop third parties having access to possibly sensitive information.

There are some similarities in terms of goal, with the divide and conquer recursion tree / Karatsuba algorithm. This algorithm splits up large number multiplication into branches to simplify the process, just as the repeated squaring algorithm does.

Question:

What is the result of the calculation 3 ^ 86, using modular exponentiation? Round to 5 decimal points.

a) 1.25832 * 10 ^ 40
b) 3.28257 * 10 ^ 41
c) 1.07753 * 10 ^ 41
d) 3.28257 * 10 ^ 20

Answer:
Convert 86 to binary = 1010110
Initialise the result to 1

First digit of binary is 1, so Result * 3 = 3.
3^2 = 9, value of binary is 0, so no multiplication.
9^2 = 81, value of binary is 1, so Result * 3 = 243.
243^2 = 59049, binary value is 0, no multiplication.
54049^2 = 3486784401, binary is 1, Result * 3 = 10460353203.
Result ^2 = 1.09418989 * 10^20, binary is 1, Result * 3 = 3.28256967 * 10^20.
Result ^2 = 1.07752637 * 10^41, binary is 0, so no multiplication.

Result = 1.07753 * 10^41

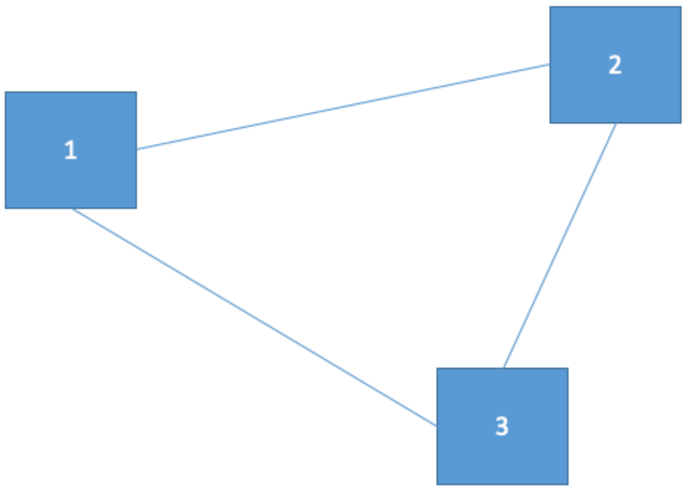
Algorithms essay

Data Structure – The Disjoint Set

The disjoint set structure keeps track of elements grouped into a number of separate sets. The algorithm performs two main functions, FIND, which is used to find what subset a given value is found in. It can also find if two elements are found within the same subset. The second main function is UNION, which joins two subsets together.

At the most basic level, this data structure could be built using nested arrays, each nested array containing a subset of values in the greater set. Find would be used to search each array and returning the index of the array it was found in. Union could be implemented by creating a new array with a size n + m (if n and m are the size of array 1 and 2), then assigning each index to the values of the two arrays. Vectors could also be used and would be able to use the push_back function to remove the need to create a new array. Alternatively a parent array is made, which stores all the parent nodes and then each parent has it own array containing its children.

The Disjoint set can be and is used during Kruskal’s algorithm, to search for loops while trying to place the next edge.



All initialized to -1

1 = -1
2 = -1
3 = -1

Go to the first edge 1-2
Find the subset 1 and 2 are in, they are in
Different sets, so take the union. 1 is made
Parent of 2. 1 represents subset 1,2.

1 = 2
2 = -1
3 = -1

Edge 2-3, 2 is in subset 2 and 3 is in subset 3,
So call union. 2 is made parent of 3. 2 now
Represents subset 1,2,3.

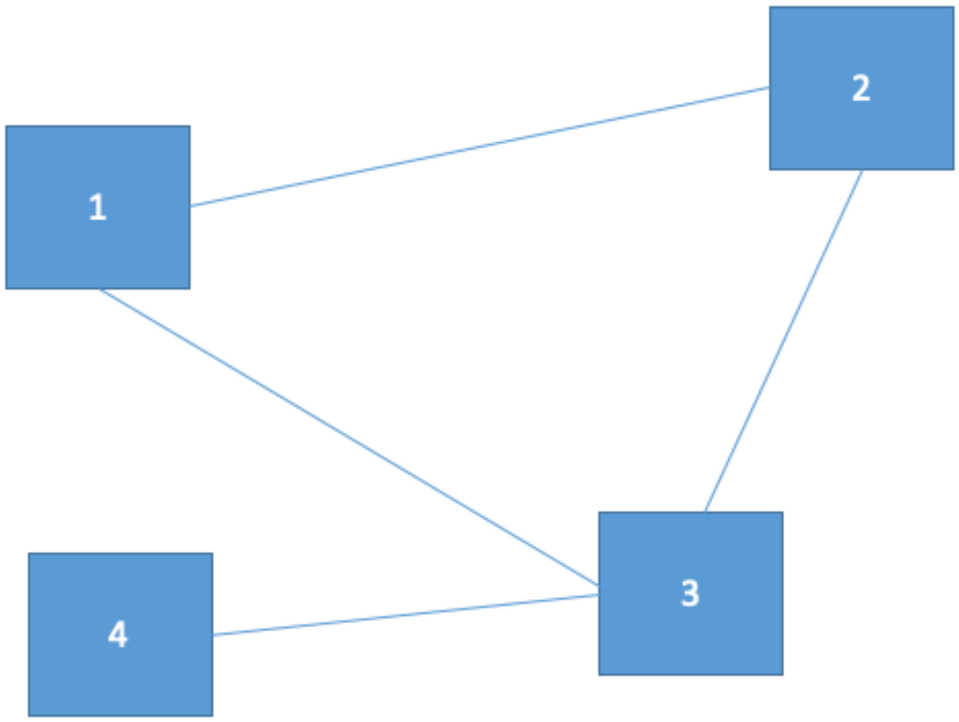
1 = 1
2 = 3
3 = -1

Edge 1-3, 1 is in subset 2 and 3 is in subset 2.
This means a cycle is present.

Question:

If we have a graph as shown below, what will the resulting values of the nodes be?
(Not multiple choice)

Algorithms essay



Answer:
Initialise all to -1, all are members of themselves
1 = -1; 2 = -1; 3 = -1; 4 = -1
Edge 1-2: 1 = {1,2}
Edge 2-3: 2 = {1,2,3}
Edge 3-1: 3 and 1 are in the same set, a loop is found
Edge 3-4: 3 = {3,4}

Bibliography:

[1]"Exponentiation by squaring", *En.wikipedia.org*, 2019. [Online]. Available: https://en.wikipedia.org/wiki/Exponentiation_by_squaring. [Accessed: 18- Mar- 2019].

[2]"Disjoint-set data structure", *En.wikipedia.org*, 2019. [Online]. Available: https://en.wikipedia.org/wiki/Disjoint-set_data_structure#MakeSet. [Accessed: 18- Mar- 2019].

[3]"Disjoint Set (Or Union-Find) | Set 1 (Detect Cycle in an Undirected Graph) - GeeksforGeeks", *GeeksforGeeks*, 2019. [Online]. Available: https://www.geeksforgeeks.org/union-find/. [Accessed: 19- Mar- 2019].