CS 58000\_01/02I Design, Analysis, and Implementation Algorithms (3 cr.)

Assignment As\_02 (Exam 01)

Student Name: Truc L. Huynh

Problem 1[30 points]: 140-30= 110 So far, you have an A for the course.

In Ch 00\_03, we addressed Figure 1.4 Modular Exponentiation: Given a function modexp(x, y, N) for computing xy mod N, where x, y, and N are integers. We also addressed when k is a power of 2, and a is any integer. We also addressed Fermat’s Little Theorem.

1a. What is (mod 17)?

**Answer**

My approach will be Modular exponential and repeated squaring

Convert 2018 to base 2, we have 11111100010. The number 11111100010 is the sum of:

10000000000 = 1024

1000000000 = 512

100000000 = 256

10000000 = 128

1000000 = 64

100000 = 32

10 = 2

Therefore, 4^2^2018 can be written as 4^2^(1024+512+256+128+64+32+2). Thus, it can further develop as 4^2^1024\* 4^2^ 512\* 4^2^256\*4^2^128\* 4^2^64\* 4^2^32\* 4^2^2. Therefore, we have

4^2^2018 (mod 17)

= (4^2^1024 \* 4^2^512 \* 4^2^256 \* 4^2^128 \* 4^2^64 \* 4^2^32 \* 4^2^2) mod 17

= [(4^2^1024 mod 17) \* (4^2^512 mod 17) \* (4^2^256 mod 17) \* (4^2^128 mod 17) \* (4^2^64 mod 17) \* (4^2^32 mod 17) \* (4^2^2 mod 17)] mod 17

Using divide and conquer, I will calculate each module's section separately. For example, let us calculate 4^2^1024 mod 17:

4^2^1024 mod 17

= (4^2^512 \* 4^2^512) mod 17

= (4^2^256 \* 4^2^256 \* 4^2^256 \* 4^2^256) mod 17

= (4^2^256 \* 4^2^256 \* 4^2^256 \* 4^2^256 \* 4^2^256 \* 4^2^256 \* 4^2^256 \* 4^2^256) mod 17

= 4^2^64 multiple with itself 16 times then mod 17

= 4^2^32 multiple with itself 32 times then mod 17

= 4^2^16 multiple with itself 64 times then mod 17

= 4^2^8 multiple with itself 128 times then mod 17

= 4^2^4 multiple with itself 256 times then mod 17

= 4^2^2 multiple with itself 512 times then mod 17

The result would be (4^2^2 mod 17) multiply with itself 512 times and mod 17 can be written as

= [(4^2^2 mod 17) \* (4^2^2 mod 17) …\* (4^2^2 mod 17) \* (4^2^2 mod 17)] mod 17 (1)

(\*) Apply the fact that (X^a)2 = X^2a and X^(a+b) = X^a\*X^b. I will double both the Number Columns and the result columns for each iteration to calculate the

|  |  |  |
| --- | --- | --- |
| number | Result mod | Result |
| 4^1 mod 17 | 4 mod 17 |  |
| 4^2 mod 17 | 16 mod 17 |  |
| 4^4 mod 17 | 256 mod 17 | 1 |

Therefore, 4^2^2 mod 17 equals 1. replace this result with (1), and we got 4^2^2 time itself 512 times (equals 1 time itself 512 times) then mod 17. We have the result is 1 mod 17 = 1. Therefore, 4^2^1024 mod 17 = 1

Applying the same method, I found:

4^2^512 mod 17 = 1

4^2^256 mod 17 = 1

4^2^128 mod 17 = 1

4^2^64 mod 17 = 1

4^2^32 mod 17 = 1

4^2^2 mod 17 = 1

Therefore, [(4^2^1024 mod 17) \* (4^2^512 mod 17 ) \* (4^2^256 mod 17) \* (4^2^128 mod 17) \* (4^2^64 mod 17) \* (4^2^32 mod 17) \* (4^2^2 mod 17)] mod 17 = 1 mod 17 = 1.

4^2^2018 (mod 17) = 1

The above way in not the one I would like to have: See my soln:

(mod 17) (mod 17)

(mod 17)

(mod 17)

(mod 17)

(mod 17)

(mod 17)

1 (mod17)

= 1

1b. What is (𝑚𝑜𝑑 31)? (Hard Problem) The answer is 8. -5

My approach will be Modular exponential and repeated squaring

Convert 2006 to base 2, we have 11111010110. The number 11111010110 is the sum of:

10000000000 = 1024

1000000000 = 512

100000000 = 256

10000000 = 128

1000000 = 64

10000 = 16

100 = 4

10 = 2

Therefore, 4^2^2006 can be written as 4^2^(1024+512+256+128+64+16+4+2). Thus, it can further develop as 4^2^1024 \* 4^2^512 \* 4^2^256 \* 4^2^128 \* 4^2^64 \* 4^2^16 \* 4^2^4 \* 4^2^2 \* Therefore, we have:

4^2^2006 (mod 31)

= (4^2^1024 \* 4^2^512 \* 4^2^256 \* 4^2^128 \* 4^2^64 \* 4^2^16 \* 4^2^4 \* 4^2^2) mod 31

= [(4^2^1024 mod 31) \* (4^2^512 mod 31) \* (4^2^256 mod 31) \* (4^2^128 mod 31) \* (4^2^64 mod 31) \* (4^2^16 mod 31) \* (4^2^4 mod 31) \* (4^2^2 mod 31)] mod 31

Using divide and conquer, I will calculate each module's section separately. For example, let us calculate 4^2^1024 mod 17:

4^2^1024 mod 31

= (4^2^512 \* 4^2^512) mod 31

= (4^2^256 \* 4^2^256 \* 4^2^256 \* 4^2^256) mod 31

= (4^2^256 \* 4^2^256 \* 4^2^256 \* 4^2^256 \* 4^2^256 \* 4^2^256 \* 4^2^256 \* 4^2^256) mod 31

= 4^2^64 multiple with itself 16 times then mod 31

= 4^2^32 multiple with itself 32 times then mod 31

= 4^2^16 multiple with itself 64 times then mod 31

= 4^2^8 multiple with itself 128 times then mod 31

= 4^2^4 multiple with itself 256 times then mod 31

= 4^2^2 multiple with itself 512 times the mod 31

The result would be (4^2^2 mod 31) multiply with itself 512 times and then mod 31

= [(4^2^2 mod 31) \* (4^2^2 mod 31) …\* (4^2^2 mod 31) \* (4^2^2 mod 31)] mod 31 (result-2)

(\*) Apply the fact that (X^a)^2 = X^2a and X^(a+b) = X^a\*X^b. I will double both the Number Columns and the result columns for each iteration to calculate the result (1)

|  |  |  |
| --- | --- | --- |
| number | Result mod | Result |
| 4^1 mod 31 | 4 mod 31 |  |
| 4^2 mod 31 | 16 mod 31 |  |
| 4^4 mod 31 | 256 mod 31 | 8 |

Therefore, 4^2^2 mod 31 equals 8. replace this result with result-2, we got 8 times itself 512 times then mod 31. We have the result 4^2^1024 mod 31 = 8^512 mod 31. Using the same methods in table 1 we have (2)

|  |  |  |
| --- | --- | --- |
| number | Result mod | Convert |
| 8^1 mod 31 | 8 mod 31 | 8 |
| 8^2 mod 31 | 8\*8 mod 31 | 2 |
| 8^4 mod 31 | 2\*2 mod 31 | 4 |
| 8^8 mod 31 | 4 \* 4 mod 31 | 16 |
| 8^16 mod 31 | 16 \* 16 mod 31 | 8 |
| 8^32 mod 31 | 8 \* 8 mod 31 | 2 |
| 8^64 mod 31 | 2 \* 2 mod 31 | 4 |
| 8^128 mod 31 | 4 \* 4 mod 31 | 16 |
| 8^256 mod 31 | 16 \* 16 mod 31 | 8 |
| 8^512 mod 31 | 8 \*8 mod 31 | 2 |

As a result, 4^2^1024 mod 31 = 8^512 mod 31 = 2

Apply the same from (1) and (2), and we have:

4^2^512 mod 31 = 8^256 mod 31 = 8

4^2^256 mod 31 = 8^128 mod 31 = 16

4^2^128 mod 31 = 8^64 mod 31 = 4

4^2^64 mod 31 = 8^32 mod 31 = 2

4^2^16 mod 31 = 8^8 mod 31 = 16

4^2^4 mod 31 = 8^2 mod 31 = 2

4^2^2 mod 31 = 8^1 mod 31 = 8

Therefore:

4^2^2006 (mod 31)

= [(4^2^1024 mod 31) \* (4^2^512 mod 31) \* (4^2^256 mod 31) \* (4^2^128 mod 31) \* (4^2^64 mod 31) \* (4^2^16 mod 31) \* (4^2^4 mod 31) \* (4^2^2 mod 31)] mod 31

= [(8^512 mod 31) \* (8^256 mod 31) \* (8^128 mod 31) \* (8^64 mod 31) \* (8^32 mod 31) \* (8^8 mod 31) \* (8^2 mod 31) \* (8^1 mod 31)] mod 31

= (2\*8\*16\*4\*2\*16\*2\*8)mod 31

= (16\*16\*8\*16\*16)mod 31

= (16^4\*8)mod 31

= (2^4^4\*2^3)mod31

= (2^16\*2^2\*2^1)mod31

= [(2^16 mod 31)\*(2^2mod31)\*(2mod31)]mod31

= (2\*4\*2) mod31

= 16

4^2^2006 (mod 31) = 16

|  |  |  |
| --- | --- | --- |
| number | Result mod | Convert |
| 2^1 mod 31 | 2 mod 31 | 2 |
| 2^2 mod 31 | 2\*2 mod 31 | 4 |
| 2^4 mod 31 | 4 \* 4 mod 31 | 16 |
| 2^8 mod 31 | 16 \* 16 mod 31 | 8 |
| 2^16 mod 31 | 8\*8 mod 31 | 2 |

The solution is:

= 32 ≡ 1(𝑚𝑜𝑑 31)

= 16 ≡ 1(𝑚𝑜𝑑 5)

= ≡ 1(𝑚𝑜𝑑 5)

≡ = ≡ ≡ 3 (𝑚𝑜𝑑 5)

= = = ≡ (𝑚𝑜𝑑 31) ≡8 (𝑚𝑜𝑑 31)

The answer is (𝑚𝑜𝑑 31) = 8.

Key is: = ≡ (𝑚𝑜𝑑 31) = 8

1c. Construct (Design) a polynomial-time algorithm for computing (mod p), where

x, y, z, and a prime p.

Algorithm modexp (x,y,z,p):

Input: two n-bits integers x and p (p is a prime number); two integer exponent y and z.

Output: x^y^z mod p

if (y = 0) then return 1;

y = y^z

// compute the y^z then we can use the same formula for x^y mod p

z = modexp(x, └y/2┘, N); // z = x└ y/2 ┘ mod N

if (y is even) then return z2 mod N;

else return x \* z2 mod N;

**Problem 2 [60 points]:**

This problem is an exercise using the formalization of the RSA public-key cryptosystem. For solving the problems, you are required to use the following formalization of the RSA public-key cryptosystem.

Given the following formalization of the RSA public-key cryptosystem, each participant creates their public key (n, g) where a is a small prime number, and n is the product of two large primes, p and q. However, the two large primes p and q are secret keys.

1. Select two very large prime numbers p and q. The number of bits needed to represent p and q might be 1024.
2. Compute

n = pq

(n) = (p – 1) (q – 1).

The formula for (n) is owing to the Theorem: The number of elements in is given by Euler’s totient function, which is

where the product is over all primes that divide n, including n if n is prime.

1. Choose a small prime number as an encryption component g, that is relatively prime to (n). That means,

gcd(g, (n) ) = 1, i.e.,

gcd(g, (p-1)(q-1)) = 1.

1. Compute the multiplicative inverse That is,

The inverse exists and is unique.

That is, the decryption component h = g-1 mod (n).

1. Let pkey = (n, g) be the public key, and skey = (p, q, h) be the secret key.

* For any message M mod n, the encryption of M is C = Mg mod n.
* The decryption of C is M = Ch mod n.

End of the formalization of the RSA public-key cryptosystem.

Use the RSA Cryptosystem formalism for solving problem 2.

Given g = 59, p = 991 and q = 997.

2a. Show that the given values of g, p, and q are prime.

**Answer**

The factor of 59 is 1, 59

The factor of 991 is 1, 991

The factor of 997 is 1, 997

Apply Fermat's little theorem:

* + If a = 2 and p = 59:
    - Using Fermat's theorem: a p-1 mod p = 1 mod p then p is a prime number
    - 2^58 ?= 1
    - 288,230,376,151,711,744 mod 59 ?= 1
    - 1 = 1
    - Therefore, 59 is a prime number
  + If a = 2 and p = 991:
    - Using Fermat's theorem: a p-1 mod p = 1 mod p then p is a prime number
    - 2991-1 mod 991 ?= 1 mod 991
    - 2990 mod 991 ?= 1 mod 991
    - 1.0463951242053391806136963369727e+298 mod 991 ?= 1
    - 1 = 1
    - Therefore 991 is a prime number
  + If a = 2 and p = 997:
    - Using Fermat's theorem: a p-1 mod p = 1 mod p then p is a prime number
    - 2997-1 mod 997 ?= 1 mod 997
    - 2996 mod 997 ?= 1 mod 997
    - 6.696928794914170755927656556625e+299 ?= 1
    - 1 = 1
    - Therefore 997 is a prime number

Use algorithm sieve is quicker than using Fermat’s Little Thm.

2b. Compute n = pq and (n) = (p – 1) (q – 1).

**Answer**

Given g = 59, p = 991 and q = 997. Therefore, n = pq = 988027.

(n) = (p – 1) (q – 1) = (991 –1) (997 – 1) = 986040

(n) = 986040

n = 988027

2c. Given a plaintext **M = 5065747269**, what is the encryption of M, using C = Mg mod n? Show in detail how you derive C, which is the ciphertext of the plaintext M.

**Answer**

M= 5,06,5,7,4,7,2,6,9

Given g = 59, p = 991, q = 997, n = 998027, (n) = 986040

Therefore,

C = Mg mod n

5: C = 559 mod 998027 = 610651

06: C = 659 mod 998027 = 276250

5: C = 559 mod 998027 = 610651

7: C = 759 mod 998027 = 848338

4: C = 459 mod 998027 = 50246

7: C = 759 mod 998027 = 848338

2: C = 259 mod 998027 = 380645

6: C = 659 mod 998027 = 276250

9: C = 959 mod 998027 = 212630

C = 61065127625061065184833850246848338380645276250212630

Encode: C = Me (mod pq), where e = g = 59

59 = 32 + 16 + 8 + 2 + 1

Compute C = Me (mod pq)

= 506574726959 (% 988027)

= 5065747269(32+16+8+2+1) (% 988027) =? = 433940

Compute C = Me (mod pq)

= 506574726959 (% 988027)

= 5065747269(32+16+8+2+1) (% 988027) =? = 433940

Give **M = 5065747269** and n = pq = 988027.

M mod n = 132840

M\*\*2 mod n ≡ 1328402 mod 988027 = 303380

M\*\*4 mod n ≡ 3033802 mod 988027 = 757242

M\*\*8 mod n ≡ 7572422 mod 988027 = 144736

M\*\*16 mod n ≡ 1447362 mod 988027 = 361242

M\*\*32 mod n ≡ 3612422 mod 988027 = 140485

506574726959 (mod 988027)

≡ (506574726932 \* 506574726916 \* 50657472698 \* 50657472692 \* 5065747269)(mod 988027)

≡ ((5065747269)(mod 988027) \* (50657472692)(mod 988027) \* (50657472698)(mod 988027) \* (506574726916)(mod 988027)\* (506574726932)(mod 988027)) mod 988027

≡ (132840 \* 303380 \* 144736 \* 361242 \* 140485) mod 988027

≡ (((((((132840 \* 303380)mod 988027) \* 144736) mod 988027) \*

361242) mod 988027) \* 140485) mod 988027

≡ (((((365897 \* 144736) mod 988027) \* 361242) mod 988027) \* 140485) mod 988027

≡ (((220992 \* 361242) mod 988027) \* 140485) mod 988027

≡ (986518\* 140485)mod 988027

= 433940

***The ciphertext of the plaintext 5065747269 is*** **433940**

2d. Compute the multiplicative inverse That is, the decryption component h = g-1 mod (n).

[Hints: Compute a GCD as a Linear Combination. Then, find an Inverse Modulo n. In other words, you can apply the extended Euclid algorithm to find the linear combination of g and Then find a positive inverse of g mod ]

**Answer**

We have Given g = 59, p = 991, q = 997, n = 998027, (n) = 986040

h = g-1 mod (n) = 59-1 mod (986040) = 584939

Using this equation h = ((((n) \*i)+1)/g) = (((986040\*i)+1)/59)

Increase i by 1 start with i=1 and choose the result as an integer h.

when i=1, h=16712.5593220339  
when i=2, h=33425.101694915254  
when i=3, h=50137.64406779661  
when i=4, h=66850.18644067796  
when i=5, h=83562.72881355933  
when i=6, h=100275.27118644067  
when i=7, h=116987.81355932204  
when i=8, h=133700.35593220338  
when i=9, h=150412.89830508476  
when i=10, h=167125.4406779661  
when i=11, h=183837.98305084746  
when i=12, h=200550.5254237288  
when i=13, h=217263.06779661018  
when i=14, h=233975.61016949153  
when i=15, h=250688.15254237287  
when i=16, h=267400.69491525425  
when i=17, h=284113.23728813557  
when i=18, h=300825.77966101695  
when i=19, h=317538.3220338983  
when i=20, h=334250.86440677964  
when i=21, h=350963.406779661  
when i=22, h=367675.9491525424  
when i=23, h=384388.4915254237  
when i=24, h=401101.0338983051  
when i=25, h=417813.57627118647  
when i=26, h=434526.1186440678  
when i=27, h=451238.66101694916  
when i=28, h=467951.2033898305  
when i=29, h=484663.74576271186  
when i=30, h=501376.28813559323  
when i=31, h=518088.83050847455  
when i=32, h=534801.3728813559  
when i=33, h=551513.9152542372  
when i=34, h=568226.4576271187  
when i=35, h=584939.0

Therefore, h=584939 Although ans is correct, but this is not the method for finding it. -5 Inefficient calculation.

Soln:

Find a positive inverse of 59 (mod (p-1)(q-1)) = 59 (mod 986040)

gcd(986040, 59) 986040 = 16712 \* 59 + 32

1 = -13\*59 + 24\*(986040 - 16712\*59)

1 = 24\*986040 – 401101\*59

= gcd(59, 32) 59 = 1 \* 32 + 27 1 = 11\*32 – 13\*(59 - 1\*32)

1 = -13\*59 + 24\*32

= gcd(32, 27) 32 = 1 \* 27 + 5 1 = -2\*27 + 11\*5

1 = -2\*27 + 11\*(32 - 1\*C)

1 = 11\*32 – 13\*27

= gcd(27, 5) 27 = 5 \* 5 + 2 1 = 1\*5 – 2\*2

1 = 1\*5 – 2\*(27 - 5\*5)

1 = -2\*27 + 11\*5

= gcd(5, 2) 5 = 2 \* 2 + 1 1 = 1\* 1 = 1\*(5 - 2 \* 2)

1 = 1\*5 – 2\*2

= gcd(2, 1) 2 = 2 \* 1 + 0 1 = 1 \* 1 - 0 \*(1\*2 = 2\*1)

1 = 1 \* 1

= gcd(1, 0) 1 = 0 \* 0 + 1 1 = 1 \* 1 - 0 \* 0

= 1

We have 1 = 24\*986040 - 401101\*59

1 (mod 986040)24\*986040 – 401101\*59)(mod 986040)

1 (mod 986040)(24\*986040(mod 986040)

– 401101\*59(mod 986040))(mod 986040)

1 (mod 986040)(0–401101\*59)(mod 986040)

1 (mod 986040) -401101\*59 (mod 986040)

1(mod 986040) -401101\*59 (mod 986040)

-401101\*59 (mod 986040) 1(mod 986040)

-401101 (mod 986040)

-401101 is the multiplicative inverse of 59.

-401101 + 986040 = 584939 is the smallest positive multiplicative inverse of 59

The secret key is ( 988027, **584939**)

2e. From problem 2d, what is your secret key (p, q, h)?

**Answer**

We have (p = 991, q = 997, h= 584939)

Also, n = 998027, (n) = 986040

2f. What is the decryption of C using M = Ch mod n? Show in detail how you derive M, which is the plaintext M of the ciphertext C.

**Answer**

C = 610651 276250 610651 848338 50246 848338 380645 276250 212630

We have 584939 = 10001110110011101011 which is the sum of

10000000000000000000, to decimal is: 524288  
1000000000000000, to decimal is: 32768  
100000000000000, to decimal is: 16384  
10000000000000, to decimal is: 8192  
100000000000, to decimal is: 2048  
10000000000, to decimal is: 1024  
10000000, to decimal is: 128  
1000000, to decimal is: 64  
100000, to decimal is: 32  
1000, to decimal is: 8  
10, to decimal is: 2  
1, to decimal is: 1

**Decrypt the first character 610651:** M = Ch mod n = 610651^584939 mod 988027

610651^1 mod 988027: 610651  
610651^2 mod 988027: 409650  
610651^4 mod 988027: 688658  
610651^8 mod 988027: 833072  
610651^16 mod 988027: 19871  
610651^32 mod 988027: 633868  
610651^64 mod 988027: 545685  
610651^128 mod 988027: 541965  
610651^256 mod 988027: 454530  
610651^512 mod 988027: 87173  
610651^1024 mod 988027: 216272  
610651^2048 mod 988027: 379804  
610651^4096 mod 988027: 124443  
610651^8192 mod 988027: 713078  
610651^16384 mod 988027: 42750  
610651^32768 mod 988027: 700577  
610651^65536 mod 988027: 780544  
610651^131072 mod 988027: 858899  
610651^262144 mod 988027: 96732  
610651^524288 mod 988027: 464134

M = Ch mod n

= 610651^584939 mod 988027

= (610651^524288 \* 610651^32768 \* 610651^16384 \* 610651^8192 \* 610651^2048 \* 610651^1024 \* 610651^128 \* 610651^64 \* 610651^32 \* 610651^8 \* 610651^2 \* 610651^1) mod 988027

= [(610651^524288(mod 988027)) \* (610651^32768(mod 988027)) \* (610651^16384(mod 988027)) \* (610651^8192(mod 988027)) \* (610651^2048(mod 988027)) \* (610651^1024(mod 988027)) \* (610651^128(mod 988027)) \* (610651^64(mod 988027)) \* (610651^32(mod 988027)) \* (610651^8(mod 988027)) \* (610651^2(mod 988027)) \* (610651^1(mod 988027))] mod 988027

= (464134 \* 700577 \* 42750 \* 713078 \* 379804 \* 216272 \* 541965 \* 545685 \* 633868 \* 833072 \* 409650 \* 610651) mod 988027

= 664356

The answer is not correct, but I can’t track where I do it wrong. I have been gone over this multiple times but can’t see what is wrong.

**Decrypt the character 276250:** M = Ch mod n = 276250^584939 mod 988027

276250^1 mod 988027: 276250  
276250^2 mod 988027: 833074  
276250^4 mod 988027: 388082  
276250^8 mod 988027: 707060  
276250^16 mod 988027: 85816  
276250^32 mod 988027: 620625  
276250^64 mod 988027: 968891  
276250^128 mod 988027: 616506  
276250^256 mod 988027: 481541  
276250^512 mod 988027: 690024  
276250^1024 mod 988027: 933222  
276250^2048 mod 988027: 973972  
276250^4096 mod 988027: 925652  
276250^8192 mod 988027: 778326  
276250^16384 mod 988027: 391712  
276250^32768 mod 988027: 661925  
276250^65536 mod 988027: 180367  
276250^131072 mod 988027: 477687  
276250^262144 mod 988027: 34319  
276250^524288 mod 988027: 65577

M = Ch mod n

= 276250 ^584939 mod 988027  
= (276250^524288 \* 276250^32768 \* 276250^16384 \* 276250^8192 \* 276250^2048 \* 276250^1024 \* 276250^128 \* 276250^64 \* 276250^32 \* 276250^8 \* 276250^2 \* 276250^1) mod 988027   
= [(276250^524288(mod 988027)) \* (276250^32768(mod 988027)) \* (276250^16384(mod 988027)) \* (276250^8192(mod 988027)) \* (276250^2048(mod 988027)) \* (276250^1024(mod 988027)) \* (276250^128(mod 988027)) \* (276250^64(mod 988027)) \* (276250^32(mod 988027)) \* (276250^8(mod 988027)) \* (276250^2(mod 988027)) \* (276250^1(mod 988027))] mod 988027   
= (65577 \* 661925 \* 391712 \* 778326 \* 973972 \* 933222 \* 616506 \* 968891 \* 620625 \* 707060 \* 833074 \* 276250) mod 988027

= 112625

**Decrypt the character** **848338:** M = Ch mod n = 848338 ^584939 mod 988027

848338^1 mod 988027: 848338  
848338^2 mod 988027: 471498  
848338^4 mod 988027: 336896  
848338^8 mod 988027: 301218  
848338^16 mod 988027: 776087  
848338^32 mod 988027: 880126  
848338^64 mod 988027: 703660  
848338^128 mod 988027: 508901  
848338^256 mod 988027: 566615  
848338^512 mod 988027: 100764  
848338^1024 mod 988027: 418244  
848338^2048 mod 988027: 827267  
848338^4096 mod 988027: 943388  
848338^8192 mod 988027: 777889  
848338^16384 mod 988027: 88333  
848338^32768 mod 988027: 269670  
848338^65536 mod 988027: 157619  
848338^131072 mod 988027: 798273  
848338^262144 mod 988027: 900582  
848338^524288 mod 988027: 287072

M = Ch mod n

= 848338 ^584939 mod 988027  
= (848338^524288 \* 848338^32768 \* 848338^16384 \* 848338^8192 \* 848338^2048 \* 848338^1024 \* 848338^128 \* 848338^64 \* 848338^32 \* 848338^8 \* 848338^2 \* 848338^1) mod 988027   
= [(848338^524288(mod 988027)) \* (848338^32768(mod 988027)) \* (848338^16384(mod 988027)) \* (848338^8192(mod 988027)) \* (848338^2048(mod 988027)) \* (848338^1024(mod 988027)) \* (848338^128(mod 988027)) \* (848338^64(mod 988027)) \* (848338^32(mod 988027)) \* (848338^8(mod 988027)) \* (848338^2(mod 988027)) \* (848338^1(mod 988027))] mod 988027   
= (287072 \* 269670 \* 88333 \* 777889 \* 827267 \* 418244 \* 508901 \* 703660 \* 880126 \* 301218 \* 471498 \* 848338) mod 988027

= 422920

**Decrypt the character** **50246:** M = Ch mod n = 50246 ^584939 mod 988027

50246^1 mod 988027: 50246  
50246^2 mod 988027: 251531  
50246^4 mod 988027: 523043  
50246^8 mod 988027: 171846  
50246^16 mod 988027: 896740  
50246^32 mod 988027: 296651  
50246^64 mod 988027: 226965  
50246^128 mod 988027: 347526  
50246^256 mod 988027: 864277  
50246^512 mod 988027: 632027  
50246^1024 mod 988027: 788683  
50246^2048 mod 988027: 572423  
50246^4096 mod 988027: 792703  
50246^8192 mod 988027: 778425  
50246^16384 mod 988027: 377849  
50246^32768 mod 988027: 953328  
50246^65536 mod 988027: 603715  
50246^131072 mod 988027: 497249  
50246^262144 mod 988027: 835197  
50246^524288 mod 988027: 50620

M = Ch mod n

= 50246 ^584939 mod 988027  
= (50246^524288 \* 50246^32768 \* 50246^16384 \* 50246^8192 \* 50246^2048 \* 50246^1024 \* 50246^128 \* 50246^64 \* 50246^32 \* 50246^8 \* 50246^2 \* 50246^1) mod 988027  
= [(50246^524288(mod 988027)) \* (50246^32768(mod 988027)) \* (50246^16384(mod 988027)) \* (50246^8192(mod 988027)) \* (50246^2048(mod 988027)) \* (50246^1024(mod 988027)) \* (50246^128(mod 988027)) \* (50246^64(mod 988027)) \* (50246^32(mod 988027)) \* (50246^8(mod 988027)) \* (50246^2(mod 988027)) \* (50246^1(mod 988027))] mod 988027   
= (50620 \* 953328 \* 377849 \* 778425 \* 572423 \* 788683 \* 347526 \* 226965 \* 296651 \* 171846 \* 251531 \* 50246) mod 988027

= 6097

**Decrypt the character** **380645:** M = Ch mod n = 380645 ^584939 mod 988027

380645^1 mod 988027: 380645  
380645^2 mod 988027: 408583  
380645^4 mod 988027: 61888  
380645^8 mod 988027: 531892  
380645^16 mod 988027: 412565  
380645^32 mod 988027: 491881  
380645^64 mod 988027: 842455  
380645^128 mod 988027: 4088  
380645^256 mod 988027: 903312  
380645^512 mod 988027: 591124  
380645^1024 mod 988027: 966529  
380645^2048 mod 988027: 755395  
380645^4096 mod 988027: 444553  
380645^8192 mod 988027: 233215  
380645^16384 mod 988027: 325929  
380645^32768 mod 988027: 14082  
380645^65536 mod 988027: 697324  
380645^131072 mod 988027: 308845  
380645^262144 mod 988027: 119418  
380645^524288 mod 988027: 465033

M = Ch mod n

= 380645 ^584939 mod 988027  
= (380645^524288 \* 380645^32768 \* 380645^16384 \* 380645^8192 \* 380645^2048 \* 380645^1024 \* 380645^128 \* 380645^64 \* 380645^32 \* 380645^8 \* 380645^2 \* 380645^1) mod 988027   
= [(380645^524288(mod 988027)) \* (380645^32768(mod 988027)) \* (380645^16384(mod 988027)) \* (380645^8192(mod 988027)) \* (380645^2048(mod 988027)) \* (380645^1024(mod 988027)) \* (380645^128(mod 988027)) \* (380645^64(mod 988027)) \* (380645^32(mod 988027)) \* (380645^8(mod 988027)) \* (380645^2(mod 988027)) \* (380645^1(mod 988027))] mod 988027   
= (465033 \* 14082 \* 325929 \* 233215 \* 755395 \* 966529 \* 4088 \* 842455 \* 491881 \* 531892 \* 408583 \* 380645) mod 988027   
= 138455

**Decrypt the character** **212630:** M = Ch mod n = 212630 ^584939 mod 988027

212630^1 mod 988027: 212630  
212630^2 mod 988027: 389407  
212630^4 mod 988027: 367824  
212630^8 mod 988027: 5758  
212630^16 mod 988027: 549673  
212630^32 mod 988027: 762302  
212630^64 mod 988027: 211262  
212630^128 mod 988027: 477000  
212630^256 mod 988027: 214278  
212630^512 mod 988027: 458567  
212630^1024 mod 988027: 919052  
212630^2048 mod 988027: 200620  
212630^4096 mod 988027: 116528  
212630^8192 mod 988027: 319723  
212630^16384 mod 988027: 535282  
212630^32768 mod 988027: 965578  
212630^65536 mod 988027: 63831  
212630^131072 mod 988027: 761240  
212630^262144 mod 988027: 597884  
212630^524288 mod 988027: 72937

M = Ch mod n

= 212630 ^584939 mod 988027  
= (212630^524288 \* 212630^32768 \* 212630^16384 \* 212630^8192 \* 212630^2048 \* 212630^1024 \* 212630^128 \* 212630^64 \* 212630^32 \* 212630^8 \* 212630^2 \* 212630^1) mod 988027   
= [(212630^524288(mod 988027)) \* (212630^32768(mod 988027)) \* (212630^16384(mod 988027)) \* (212630^8192(mod 988027)) \* (212630^2048(mod 988027)) \* (212630^1024(mod 988027)) \* (212630^128(mod 988027)) \* (212630^64(mod 988027)) \* (212630^32(mod 988027)) \* (212630^8(mod 988027)) \* (212630^2(mod 988027)) \* (212630^1(mod 988027))] mod 988027   
= (72937 \* 965578 \* 535282 \* 319723 \* 200620 \* 919052 \* 477000 \* 211262 \* 762302 \* 5758 \* 389407 \* 212630) mod 988027

= 38965

Doing the same thing for the rest of them we have:

the cipher text is: 610651, the original text is 664356  
the cipher text is: 276250, the original text is 112625  
the cipher text is: 610651, the original text is 664356  
the cipher text is: 848338, the original text is 422920  
the cipher text is: 50246, the original text is 6097  
the cipher text is: 848338, the original text is 422920  
the cipher text is: 380645, the original text is 138455  
the cipher text is: 276250, the original text is 112625  
the cipher text is: 212630, the original text is 38965

After all, the correct answer should be C = 610651 276250 610651 848338 50246 848338 380645 276250 212630

M = 5065747269 Method is inefficient! The conversion of C to M is not there. -5

My 2f solution is:

The secret key is ( 988027, 584939)

Decode: M = Cd (Mod pq)

M = **433940**584939 mod 988027

584939 =219 + 215 + 214 + 213 + 211 +210 +27 + 26 + 25 +23 + 2 + 1

= 1 + 2 + 8 + 32 + 64 + 128 + 1024 + 2048 + 8192 + 16384 + 32768 + 424288

Enter C **433940**, d 584939, pq 988027

**433940 mod 988027 = 433940**

**433940**2 mod 988027 = 797805

**433940**4 mod 988027 = 884490

**433940**8 mod 988027 = 805446

**433940**16 mod 988027 = 778608

**433940**32 mod 988027 = 763112

**433940**64 mod 988027 = 762852

**433940**128 mod 988027 = 762852

**433940**256 mod 988027 = 166442

**433940**512 mod 988027 = 638338

**433940**1024 mod 988027 = 223093

**433940**2048 mod 988027 = 602578

**433940**4096 mod 988027 = 323584

**433940**8192 mod 988027 = 443731

**433940**16384 mod 988027 = 215720

**433940**32768 mod 988027 = 34727

**433940**65536 mod 988027 = 571589

**433940**131072 mod 988027 = 132750

**433940**262144 mod 988027 = 112928

**433940**424288 mod 988027 = 268695

Decode: M = Cd (Mod pq)

M = **433940**584939 mod 988027

M = (433940 \* 797805 \* 805446 \* 763112 \* 762852 \* 211039 \* 223093 \* 602578 \* 443731 \* 215720 \* 34727 \* 268695) mod 988027

M = (((((((((((((((((((((((433940 \* 797805) mod 988027) \* 805446) mod 988027) \* 763112) mod 988027) \* 762852) mod 988027) \* 211039) mod 988027) \* 223093) mod 988027) \* 602578) mod 988027) \* 443731)mod 988027) \*215720)mod 988027)\* 34727) mod 988027) \* 268695) mod 988027)mod 988027

= 132840

(mod1 \* mod2) mod n = 769062

(mod1\_2 \* mod8) mod n = 312164

(mod1\_2\_8 \* mod32) mod n = 808614

(mod1\_2\_8\_32 \* mod64) mod n = 874299

(mod1\_2\_8\_32\_64 \* mod128) mod n = 108492

(mod1\_32\_64\_128 \* mod1024) mod n = 108337

(mod1\_32\_64\_128\_1024 \* mod2048) mod n = 572842

(mod1\_32\_64\_128\_2048 \* mod8192) mod n = 23266

(mod1\_128\_2048\_8192 \* mod16384) mod n = 752387

(mod1\_2048\_8192\_16384 \* mod32768) mod n = 757361

(mod1\_8192\_16384\_32768 \* mod424288) mod n = 132840

M should be

132840 + ⨽(M/988027)⨼ \* 988027

= 132840 + 5127\*988027

= 5065747269

The plaintext M for the ciphertext 433940 is:

the decryption of C using M = Ch mod n

M = **433940**584939 mod 988027

= **5065747269**

2g. (Bonus) [5 points]:

What is the message (in terms of the alphabet)?

**Answer**

If I encode a=1, b=2, c =3, d = 4, e = 5, f = 6, g = 7, h = 8, i = 9….

The message should be: efegdgbfi

What is the message (in terms of the alphabet)? Petri

+0

Problem 3 [30 points]:

Assume that we define

h1(k) = k mod 13, and

h2(k) = 1 + (k mod 11).

For open addressing, consider the following methods

**Linear Probing**

Given an ordinary hash function h: U {0, 1, 2, …, m-1}, the method of linear probing uses the hash function h (k, i) = (h1(k) + i) mod m. for i = 0, 1, 2, …, m-1.

**Quadratic Probing**

Uses a hashing function of the form h (k, i) = (h1(k) + c1i + c2i2) mod m, where h1 is an auxiliary hash function, c1 and c2 0 are auxiliary constants c13c2 = 5, and i = 0, 1, 2, …, m-1.

**Double hashing**

Uses a hashing function of the form h (k, i) = (h1(k) + i h2(k)) mod m, where h1 and h2 are auxiliary hash functions. The value of h2(k) must never be zero and should be relatively prime to m for the sequence to include all possible addresses.

Given K = {79, 69, 98, 72, 14, 50, 88, 99, 78, 65} and the size of a table is 13, with indices counting from 0, 1, 2, …, 12. Store the given K in a table with the size 13 counting the indices from 0, 1, 2, …, 12. Show the resultant table with 10 given keys for each method applied:

**3a. if linear probing is employed.**

The Resultant Table with 10 given keys is: Complete the table.

We have K = [79, 69, 98, 72, 14, 50, 88, 99, 78, 65]

After using the h1(k) = k mod 13 the key will look like this

H1= [1, 4, 7, 7, 1, 11, 10, 8, 0, 0]

* + Since index 1, 4, and 7 is empty we push 79, 69, and 98 in the table
  + 72 gives index 7 (which is not available). We increment i by 1, now we are at index 8. Index 8 is available, so we push 72 into index 8.
  + 14 gives index 1 (which is not available). We increment i by 1, now we are at index 2. Index 2 is available, so we push 14 into index 2.
  + 50, 88 give index 11, 10 accordingly. They are all available, so we push them according to the index.
  + 99 gives index 8 (which is not available). We increment i by 1, now we are at index 9. Index 9 is available, so we push 99 into index 9.
  + 78 gives an index of 0 accordingly. It is available, so we put 78 into index 0
  + 65 gives an index of 0 (which is not available). We increment i by 1, now we are at index 1 (still not available). Keep increment by 1 until it hits Index 3. Index 3 is available, so we push 65 into index 3.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 78 | 79 | 14 | 65 | 69 |  |  | 98 | 72 | 99 | 88 | 50 |  |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |

**3b. if quadratic probing is employed.**

The Resultant Table with 10 given keys is: Complete the table.

h (k, i) = (h1(k) + c1i + c2i2) mod 13

with c13c2 = 5, h1(k) = k mod 13 we have h (k, i) = (h1(k) + 5i + (5i2)/3) mod 13

We have K = [79, 69, 98, 72, 14, 50, 88, 99, 78, 65]

We have h1= [1, 4, 7, 7, 1, 11, 10, 8, 0, 0]

I calculate the combination of h(k, i) and put them in a table for easy explanation. According to the table we have h = [1,4,7,0,5,11,10,8,6,3]

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | h(k,i) | | | | | | | | | | | | |
| k | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 79 | 1 | 7 | 4 | 5 | 8 | 2 | 0 | 0 | 4 | 12 | 9 | 10 | 2 |
| 69 | 4 | 10 | 7 | 8 | 11 | 5 | 3 | 3 | 7 | 2 | 12 | 0 | 5 |
| 98 | 7 | 0 | 10 | 11 | 1 | 8 | 6 | 6 | 10 | 5 | 2 | 3 | 8 |
| 72 | 7 | 0 | 10 | 11 | 1 | 8 | 6 | 6 | 10 | 5 | 2 | 3 | 8 |
| 14 | 1 | 7 | 4 | 5 | 8 | 2 | 0 | 0 | 4 | 12 | 9 | 10 | 2 |
| 50 | 11 | 4 | 1 | 2 | 5 | 12 | 10 | 10 | 1 | 9 | 6 | 7 | 12 |
| 88 | 10 | 3 | 0 | 1 | 4 | 11 | 9 | 9 | 0 | 8 | 5 | 6 | 11 |
| 99 | 8 | 1 | 11 | 12 | 2 | 9 | 7 | 7 | 11 | 6 | 3 | 4 | 9 |
| 78 | 0 | 6 | 3 | 4 | 7 | 1 | 12 | 12 | 3 | 11 | 8 | 9 | 1 |
| 65 | 0 | 6 | 3 | 4 | 7 | 1 | 12 | 12 | 3 | 11 | 8 | 9 | 1 |

* + We have h(79,0)=1 index 1 is empty so put 79 into index 1
  + We have h(69,0)=4 index 4 is empty so put 69 into index 4
  + We have h(98,0)=7 index 7 is empty so put 98 into index 7
  + We have h(72,0)=7 index 7 is not empty so increment I by 1. h(72,1)=0 index 0 is empty so put 72 into index 0
  + We have h(14,0)=1 index 1 is not empty so increment I by 1. We have h(14,1)=7 index 7 is not empty so increment I by 1. We have h(14,2)=4 index 4 is not empty so increment I by 1. h(14,3)=5 index 5 is empty so put 14 into index 5
  + We have h(50,0)=11 index 11 is empty so put 50 into index 11
  + We have h(88,0)=10 index 10 is empty so put 88 into index 10
  + We have h(99,0)=8 index 8 is empty so put 99 into index 8
  + We have h(78,0)=0 index 0 is not empty so increment I by 1. h(78,1)=6 index 6 is empty so put 78 into index 6
  + We have h(65,0)=0 index 0 is not empty so increment I by 1. h(65,1)=6 index 6 is not empty so increment I by 1. h(65,2)=3 index 3 is empty so put 65 into index 3

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 72 | 79 |  | 65 | 69 | 14 | 78 | 98 | 99 |  | 88 | 50 |  |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |

-5

Hash position = (h1(k) + c1i+c2i2) mod 13

Where h1(k) = k mod 13

C1 = 3, c2 = 5

k = {79, 69, 98, 72, 14, 50, 88, 99, 78, 65} This is a set of keys. We take it one by one from the beginning.

QUADRATIC PROBING

h(k, i) = (h(k) + c1i + c2i2) mod m

h(k, i) = (h(k) + 3i + 5i2) mod 13

1. h (79, 0) = (79 mod 13 + 0 + 0) mod 13 = 1 mod 13 = **1**
2. h (69, 0) = (69 mod 13 + 0 + 0) mod 13 = **4**
3. h (98, 0) = (98 mod 13 + 0+ 0) mod 13 = **7**
4. h (72, 0) = (72 mod 13 + 0+ 0) mod 13 = 7 🡪 occupied

h (72, 1) = (72 mod 13 + 3(1) + 5(1)) mod 13 = (7+3+5) mod 13 = **2**

1. h (14, 0) = (14 mod 13 + 0 + 0) mod 13 = 1 🡪 occupied

h (14, 1) = (14 mod 13 + 3 +5) mod 13 = 9 mod 13 = **9**

1. h (50, 0) = (50 mod 13 + 0 + 0) mod 13 = **11**
2. h (88, 0) = (88 mod 13 + 0 + 0) mod 13 = **10**
3. h (99, 0) = (99 mod 13 + 0 + 0) mod 13 = **8**
4. h (78, 0) = (78 mod 13 + 0 + 0) mod 13 = **0**
5. h (65, 0) = (65 mod 13 + 0) mod 13 = 0 🡪 occupied  
   h (65, 1) = (65 mod 13 + 3 +5) mod 13 = 8 mod 13 = 8 🡪 occupied  
   h (65, 2) = (65 mod 13 + 3(2) + 5(4)) mod 13 = 26 mod 13 = 0 🡪 occupied  
   h (65, 3) = (65 mod 13 + 3(3) + 5(9)) mod 13 = 54 mod 13 = 2 🡪 occupied  
   h (65, 4) = (65 mod 13 + 3(4) + 5(16)) mod 13 = 92 mod 13 = 1 🡪 occupied  
   h (65, 5) = (65 mod 13 + 3(5) + 5(25)) mod 13 = 140 mod 13 = 10 🡪 occupied  
   h (65, 6) = (65 mod 13 + 3(6) + 5(36)) mod 13 = 188 mod 13 = **3**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 78 | 79 | 72 | 65 | 69 |  |  | 98 | 99 | 14 | 88 | 50 |  |

**3c. if double hashing is employed.**

The Resultant Table with 10 given keys is: Complete the table.

We have K = [79, 69, 98, 72, 14, 50, 88, 99, 78, 65]

We have h1= [1, 4, 7, 7, 1, 11, 10, 8, 0, 0]

We have h2= [3, 4, 11, 7, 4, 7, 1, 1, 2, 11]

I calculate the combination of h(k, i) and put them in a table for easy explanation. According to the table we have h = [1,4,7,8,5,11,10,9,0,3]

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | h(k,i) | | | | | | | | | | | | |
| k | h1 | h2 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 79 | 1 | 3 | 1 | 4 | 7 | 10 | 0 | 3 | 6 | 9 | 12 | 2 | 5 | 8 | 11 |
| 69 | 4 | 4 | 4 | 8 | 12 | 3 | 7 | 11 | 2 | 6 | 10 | 1 | 5 | 9 | 0 |
| 98 | 7 | 11 | 7 | 5 | 3 | 1 | 12 | 10 | 8 | 6 | 4 | 2 | 0 | 11 | 9 |
| 72 | 7 | 7 | 7 | 1 | 8 | 2 | 9 | 3 | 10 | 4 | 11 | 5 | 12 | 6 | 0 |
| 14 | 1 | 4 | 1 | 5 | 9 | 0 | 4 | 8 | 12 | 3 | 7 | 11 | 2 | 6 | 10 |
| 50 | 11 | 7 | 11 | 5 | 12 | 6 | 0 | 7 | 1 | 8 | 2 | 9 | 3 | 10 | 4 |
| 88 | 10 | 1 | 10 | 11 | 12 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 99 | 8 | 1 | 8 | 9 | 10 | 11 | 12 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 78 | 0 | 2 | 0 | 2 | 4 | 6 | 8 | 10 | 12 | 1 | 3 | 5 | 7 | 9 | 11 |
| 65 | 0 | 11 | 0 | 11 | 9 | 7 | 5 | 3 | 1 | 12 | 10 | 8 | 6 | 4 | 2 |

I will fill the number in according to the table

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 78 | 79 |  | 65 | 69 | 14 |  | 98 | 72 | 99 | 88 | 50 |  |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |

**Problem 4 [20 points]:**

For the division method for creating hash functions, map a key k into one of the m slots by taking the remainder of k divided by m. The hash function is:

h(k) = k mod m,

where m should not be a power of 2.

For the multiplication method for creating hash functions, the hash function is

h(k) = └ m (kA –└ k A ┘) ┘ = └ m (k A mod 1) ┘

where “k A mod 1” means the fractional part of k A and a constant A in the range

0 < A < 1.

An advantage of the multiplication method is that the value of m is not critical.

Choose m = 2p for some integer p.

Give your explanations for the following questions:

4a. Why m should not be a power of 2 in the division method for creating a hash

function?

**Answer:**

m should not be a power of 2, since if m = 2^p, then h(k) is just the p lowest-order bits. According to (R-1), Prime numbers that are too close to a power of 2 will provide the same kind of biasing as a power of 2 for the keys which differ by +a or −a if 2^k = a mod M.

In cases when keys are distributed in such a way that many different keys have the same low-order p-bit patterns, then choosing m equal to a power of 2 will fail to provide uniform distribution.

-5

Soln:

4a. 2k has k-bit representation. That is, 2k = 1 is followed by a k-1 number of 0 in bits representation, 10…00. For example. 24  = 1000, 25  = 10000, 26  = 100000, etc.

Consider a key K, then h(K) = K mod 2k,and h(K) has the rightmost k-1 bit representation as its remainder to be the hash. The reason is: Let K be i+k bits long in bit representation.

Consider Ck(K) = { ui mk vi mk | ui and vi are the leftmost i-bit representations, where ui vi ; and mk is rightmost k-bit representation.} Therefore, Ck (K) has 2k-1  keys and h(K1) = K1 mod 2k = h(K2) = K2 mod 2k for K1, K2 Ck (K). For example, for C4(K) = { 1000101, 1010101, 1100101, 1110101, 1001101, 1011101, 1101101, 1111101}. Since h(K) = K mod 24 = 1012 = 5, all these 8 distinct numbers will be hashed to the same value 5.

In summary, h(K) = K mod 2k will retain the rightmost k-1 bit representation from a list of numbers Ck(K). If Ck(K). has a list of distinct numbers which have the same value of rightmost k-1 bit representation, they will have the same hashed value (namely, they collide).

**Thus, the value of m should not be a power of 2 for the division method for creating a hash function,** because if m was a power of 2, h would become the k-1 lowest-order bits of key K. And there is a list of numbers Ck(K) where each distinct K, h(K) = K mod 2k  has the same k-1 lowest-order bits. These keys contend the same location of a given table. i.e., collisions occur. mod 2k  does not provide a uniform distribution of data.

4b. Why m = 2p, for some integer p, could be (and in fact, favorably) used in the

multiplication method?

According to (R2), the multiplication method is suitable m = 2p when the table size is a power of two, then getting the index from the hash could be implemented as a bitwise AND operation. Therefore, the whole path of computing the table index by the key with multiplication hashing is very fast.

The multiplication method seems to be better in all respects than the division method including the fact that the multiplication method is not restricted to choosing prime m

-5

For the multiplication method for creating hash functions.

The hash function is

h(k)=└ m (k A mod 1) ┘

where “kA mod 1” is the fractional part of kA, and a constant A, 0 < A < 1.

For Example:

Consider that a good value for A is 0.618033

So, A = 0.618033

Consider K = 123456

Consider m = 2psuch asm=104

h(k) = └2p(123456\*0.618033 mod1) ┘

= └104 (76300.00411 mod 1) ┘

= └104 (0.0041151) ┘

= └41.151┘ = 41

Extend this notion to 2p which has a p-bit representation. That is, 2p = 1 follows by k-1 number of 0 in bits representation, 10…00. For example 24  = 1000, 25  = 10000, 26  = 100000, etc. For └ m (k A mod 1) ┘, multiplying the fraction part of kA by m is to shift left the fraction part by k-1 bits. (Is this true?)

**Thus, m=2p for some integer p could be used for the multiplication method for creating hash functions.** And in multiplication the value of m does not matter therefore, it is a power of 2 that does not matter.

Unlike the division method, m is not a critical value here, so we don’t need to avoid any value of m. We often set m to be a power of 2 (say m = 2p) for some integer p since it makes computation easier. For example: Suppose the word size of our computer is w bits. If we further restrict A to be a real of the form s/2w for some integer s, then we can compute the hash value by following these steps:

Step 1: Obtain k\*s as a 2w-bit integer

Step 2: Retain the last w bits of k\*s

Step 3: Retain the first p bits of the result of step 2

References:

[Hash size: Are prime numbers "near" powers of two a poor choice for the modulus? - Computer Science Stack Exchange](https://cs.stackexchange.com/questions/86237/hash-size-are-prime-numbers-near-powers-of-two-a-poor-choice-for-the-modulus/86345#86345) (R1)

[algorithm - What are the disadvantages of hashing function using multiplication method - Stack Overflow (R2)](https://stackoverflow.com/questions/25217770/what-are-the-disadvantages-of-hashing-function-using-multiplication-method#:~:text=Multiplication%20method%20is%20suitable%20when%20the%20table%20size,the%20key%2C%20with%20multiplication%20hashing%2C%20is%20very%20fast.)

**Note: If you provide your answer in your handwriting, good handwriting is required.**

**Proper numbering of your answer to each problem is strictly required. The problem’s solution must be given orderly. (10 points off if not)**