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 x^y mod N, where x, y, and N are integers. We also addressed $a^k \ mod \ n$, when k is a power of 2, and a is any integer. We also addressed Fermat's Little Theorem.

1a. What is $4^{2^{2018}}$ (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:

$$4^{2^{2018}} \pmod{17} \equiv 4^{2^2 * 2^{2016}} \pmod{17}$$
$$\equiv (4^{2^2})^{2^{2016}} \pmod{17}$$
$$\equiv (4^4)^{2^{2016}} \pmod{17}$$
$$\equiv (256)^{2^{2016}} \pmod{17}$$
$$\equiv (256\ (mod\ 17))^{2^{2016}} \pmod{17}$$
$$\equiv (1)^{2^{2016}} \pmod{17}$$
$$\equiv 1 \pmod{17}$$
$$= 1$$

1b. What is $4^{2^{2006}} (mod\ 31)$? (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:

$2^5 = 32 \equiv 1 (mod\ 31)$

$2^4 = 16 \equiv 1 (mod\ 5)$

$2^{2000} = (2^4)^{500} \equiv 1 (mod\ 5)$

$2^{2007} \equiv 2^7 = 2^{4+3} \equiv 2^3 \equiv 3\ (mod\ 5)$

$4^{2^{2006}} = 2^{2*2^{2006}} = 2^{2^{2007}} = 2^{5N+3} \equiv 2^3\ (mod\ 31) \equiv 8\ (mod\ 31)$

The answer is $4^{2^{2006}}\ (mod\ 31) = 8.$

Key is: $2^{2^{2007}} = 2^{5N+3} \equiv 2^3\ (mod\ 31) = 8$

1c. Construct (Design) a polynomial-time algorithm for computing x^{y^z} (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

   φ(n) = (p – 1) (q – 1).

The formula for φ(n) is owing to the Theorem: The number of elements in $z_n^* = \{ [1]_n, [2]_n, \ldots, [n-1]_n \}$ is given by Euler's totient function, which is

$$\varphi(n) = n \prod_{p:p|n} \left(1 - \frac{1}{p} \right),$$

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

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

   gcd(g, φ(n)) = 1, i.e.,

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

4. Compute the multiplicative inverse $[h]_{\varphi(n)}$ of $[g]_{\varphi(n)}$. That is,

$$[g]_{\varphi(n)}[h]_{\varphi(n)} = [1]_{\varphi(n)}.$$

The inverse exists and is unique.

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

5. 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 = M^g mod n.
   - The decryption of C is M = C^h 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
  - 2^{991-1} mod 991 ?= 1 mod 991
  - 2^{990} 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
  - 2^{997-1} mod 997 ?= 1 mod 997
  - 2^{996} mod 997 ?= 1 mod 997
  - 6.6969287949141707559276565566625e+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 φ(n) = (p – 1) (q – 1).

Answer

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

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

φ(n) = 986040

n = 988027

2c. Given a plaintext **M = 5065747269**, what is the encryption of M, using $C = M^g \bmod n$? 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, $\varphi(n) = 986040$

Therefore,

$C = M^g \bmod n$

5: $C = 5^{59} \bmod 998027 = 610651$

06: $C = 6^{59} \bmod 998027 = 276250$

5: $C = 5^{59} \bmod 998027 = 610651$

7: $C = 7^{59} \bmod 998027 = 848338$

4: $C = 4^{59} \bmod 998027 = 50246$

7: $C = 7^{59} \bmod 998027 = 848338$

2: $C = 2^{59} \bmod 998027 = 380645$

6: $C = 6^{59} \bmod 998027 = 276250$

9: $C = 9^{59} \bmod 998027 = 212630$

C = 610651276250610651848338502468483383806452762502126 30

Encode: $C = M^e \pmod{pq}$, where e = g = 59

59 = 32 + 16 + 8 + 2 + 1

Compute $C = M^e \pmod{pq}$

$= 5065747269^{59}$ (% 988027)

$= 5065747269^{(32+16+8+2+1)}$ (% 988027) =? = 433940

Compute $C = M^e \pmod{pq}$

$= 5065747269^{59}$ (% 988027)

$= 5065747269^{(32+16+8+2+1)}$ (% 988027) =? = 433940

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

M mod n = 132840

$M^{**}2 \bmod n \equiv 132840^2 \bmod 988027 = 303380$

$M^{**}4 \bmod n \equiv 303380^2 \bmod 988027 = 757242$

$M^{**}8 \bmod n \equiv 757242^2 \bmod 988027 = 144736$

$M^{**}16 \bmod n \equiv 144736^2 \bmod 988027 = 361242$

$M^{**}32 \bmod n \equiv 361242^2 \bmod 988027 = 140485$

$5065747269^{59} \pmod{988027}$

$\equiv (5065747269^{32} * 5065747269^{16} * 5065747269^{8} * 5065747269^{2} * 5065747269)(\bmod 988027)$

$\equiv ((5065747269)(\bmod 988027) * (5065747269^{2})(\bmod 988027) * (5065747269^{8})(\bmod 988027) * (5065747269^{16})(\bmod 988027) * (5065747269^{32})(\bmod 988027)) \bmod 988027$

$\equiv (132840 * 303380 * 144736 * 361242 * 140485) \bmod 988027$

$\equiv ((((((132840 * 303380) \bmod 988027) * 144736) \bmod 988027) * 361242) \bmod 988027) * 140485) \bmod 988027$

$\equiv ((((365897 * 144736) \bmod 988027) * 361242) \bmod 988027) * 140485) \bmod 988027$

$\equiv (((220992 * 361242) \bmod 988027) * 140485) \bmod 988027$

$\equiv (986518 * 140485) \bmod 988027$

$= 433940$

The ciphertext of the plaintext 5065747269 is 433940

2d. Compute the multiplicative inverse $[h]_{\varphi(n)}$ of $[g]_{\varphi(n)}$. That is, the decryption component h = g^{-1} mod φ(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 φ (n). Then find a positive inverse of g mod φ (n).]

Answer

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

h = g^{-1} mod φ(n) = 59^{-1} mod (986040) = 584939

Using this equation h = (((φ(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≡ $\frac{1}{59}$(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, φ(n) = 986040

2f. What is the decryption of C using M = C^h 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 = C^h 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 = C^h 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 = C^h 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 = C^h 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 = C^h 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 = C^h 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 = C^h 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 = C^h 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 = C^h 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 = C^h 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 = C^h 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 = C^h 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 = C^d (Mod pq)

M = **433940**^{584939} mod 988027

$584939 = 2^{19} + 2^{15} + 2^{14} + 2^{13} + 2^{11} + 2^{10} + 2^{7} + 2^{6} + 2^{5} + 2^{3} + 2 + 1$

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

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: $M = C^{d}$ (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 + $\lfloor$(M/988027)$\rfloor$ * 988027

= 132840 + 5127*988027

= 5065747269

The plaintext M for the ciphertext 433940 is:

the decryption of C using M = C^h 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

$h_1(k)$ = k mod 13, and

$h_2(k)$ = 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) = ($h_1(k) + c_1 i + c_2 i^{2)}$ mod m, where h_1 is an auxiliary hash function, c_1 and $c_2 \neq 0$ are auxiliary constants c_1=3c_2 = 5, and i = 0, 1, 2, …, m-1.

Double hashing

Uses a hashing function of the form h (k, i) = ($h_1(k)$ + i $h_2(k)$) mod m, where h_1 and h_2 are auxiliary hash functions. The value of $h_2(k)$ 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 $h_1(k) = k \bmod 13$ 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.

$h(k, i) = (h_1(k) + c_1 i + c_2 i^{2)}$ mod 13

with $c_1 3 c_2 = 5$, $h_1(k) = k \bmod 13$ we have $h(k, i) = (h_1(k) + 5i + (5i^{2)}/3) \bmod 13$

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 = $(h_1(k) + c_1 i + c_2 i^2) \bmod 13$

Where $h_1(k) = k \bmod 13$

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

$h(k, i) = (h(k) + c_1 i + c_2 i^2) \bmod m$

$h(k, i) = (h(k) + 3i + 5i^2) \bmod 13$

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**
5. 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**
6. h (50, 0) = (50 mod 13 + 0 + 0) mod 13 = **11**
7. h (88, 0) = (88 mod 13 + 0 + 0) mod 13 = **10**
8. h (99, 0) = (99 mod 13 + 0 + 0) mod 13 = **8**
9. h (78, 0) = (78 mod 13 + 0 + 0) mod 13 = **0**
10. 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:

$$h(k) = k \bmod m,$$

where m should not be a power of 2.

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

$$h(k) = \lfloor m (kA - \lfloor k A \rfloor) \rfloor = \lfloor m (k A \bmod 1) \rfloor$$

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 $m = 2^p$ 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. 2^k has k-bit representation. That is, 2^k = 1 is followed by a k-1 number of 0 in bits representation, 10…00. For example. 2^4 = 1000, 2^5 = 10000, 2^6 = 100000, etc.

Consider a key K, then $h(K) = K \bmod 2^k$, 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 $\mathcal{C}_k(K) = \{ u_i\, m_k \neq v_i\, m_k \mid u_i$ and v_i are the leftmost i-bit representations, where $u_i \neq v_i$; and m_k is rightmost k-bit representation.$\}$ Therefore, $\mathcal{C}_k(K)$ has 2^{k-1} keys and $h(K_1) = K_1 \bmod 2^k = h(K_2) = K_2 \bmod 2^k$ for $K_1, K_2 \in \mathcal{C}_k(K)$. For example, for $\mathcal{C}_4(K)$ = { 1000101, 1010101, 1100101, 1110101, 1001101, 1011101, 1101101, 1111101}. Since $h(K) = K \bmod 2^4 = 101_2 = 5$, all these 8 distinct numbers will be hashed to the same value 5.

In summary, $h(K) = K \bmod 2^k$ will retain the rightmost k-1 bit representation from a list of numbers $\mathcal{C}_k(K)$. If $\mathcal{C}_k(K)$. 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 $\mathcal{C}_k(K)$ where each distinct K, $h(K) = K \bmod 2^k$ has the same k-1 lowest-order bits. These keys contend the same location of a given table. i.e., collisions occur. mod 2^k does not provide a uniform distribution of data.

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

According to (R2), the multiplication method is suitable $m = 2^p$ 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

$$h(k)=\lfloor m\,(k\,A \bmod 1)\rfloor$$

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 $m = 2^p$ such as $m=10^4$

$h(k) = \lfloor 2^p\,(123456*0.618033 \bmod 1)\rfloor$

$= \lfloor 10^4\,(76300.00411 \bmod 1)\rfloor$

$= \lfloor 10^4\,(0.0041151)\rfloor$

$= \lfloor 41.151\rfloor = 41$

Extend this notion to 2^p which has a p-bit representation. That is, $2^p =$ 1 follows by k-1 number of 0 in bits representation, 10…00. For example 2^4 = 1000, 2^5 = 10000, 2^6 = 100000, etc. For $\lfloor m\,(k\,A \bmod 1)\rfloor$, multiplying the fraction part of kA by m is to shift left the fraction part by k-1 bits. (Is this true?)

Thus, $m=2^p$ 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 $m = 2^p$) 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]() (R1)

[algorithm - What are the disadvantages of hashing function using multiplication method - Stack Overflow (R2)]()

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)

CS 58000_01/02I Design, Analysis and Implementation Algorithms
Assignment As_03 (Exam 02)

Problem 1[30 points]:

1a. Show that for any real constants a < 0 and b > 0,

$$(n + a)^b = \Theta(n^b).$$

ANSWER:

To prove (n+a)^b = O(n^b), we must prove that there is exist constant c_1, c_2, $n_0 > 0$ such that 0 < c_1*n^b <= (n+a)^b <= c_2 * n^b for all n>= n0

n+a <= 2*n, when |a| <= n

n+a>= ½*n, when |a| <= n/2

Therefore n >= 2|a| and 0<= n/2 <= (n+a) <= 2n

As b>0, we raise all the terms of the previous inequality to the power of b without breaking the inequality.

0^b <= (n/2)^b <= (n+a)^b <= (2n)^b

0 <= (1/2)^b * n^b <= (n+a)^b <= 2^b*n^b

0 <= 1/(2^b)*n^b <= (n+a)^b <= 2^b*n^b

Therefore there is exists c_1 = 1/(2^b), c_2 = 2^b, and n_0 = 2|a|

1b. Explain why the statement "The running time of algorithm A is at least O(n2)" is meaningless.

ANSWER:

T(n): running time of Algo A.

T(n) The statement is: T(n) >= O(n^2). To decide the meaning of a function we need to decide the upper bound and lower bound of it.

Upper bound: Because T(n) >= O(n^2), then there's no information about upper bound of T(n)

Lower bound: Assume f(n) = O(n^2), then the statement is: T(n) >= f(n), but f(n) could be anything that is "smaller" than n^2. Ex: constant, n,..., So there's no conclusion about lower bound of T(n) too

Therefore, The statement is meaningless

Reference:

big o - Running time of algorithm A is at least O(n^2) - Why is it meaningless? - Stack Overflow

1c. Is 2n+1 = O(2n)? Justify your answer.

Is 22n = O(2n)? Justify your answer.

ANSWER:

Here

Is $2^{n+1} = O(2^n)$? Justify your answer

n	2^(n+1)	2^n	compare
0	2	1	not equal
1	4	2	not equal
2	8	4	not equal

Therefore 2^n+1 is not equal to O(2^n) and
Is $2^{2n} = O(2^n)$? Justify your answer.

n	2^(n+1)	2^n	compare
0	1	1	equal
1	4	2	not equal

Therefore, 2^{2n} is not equal $O(2^n)$

Problem 2 [30 points]:

Order of the following functions according to their order of growth (from the lowest to the highest)

(n – 2)!, 22n, 0.002 n4 + 3n2 +1, 2^n, e^n, $n2^n$, 1n2n, 3√ n, 3n, $2^{\log n}$, n^2, $4^{\log n}$, $\sqrt{\log n}$

{Hint: 1n2 n = (loge n) (loge n) where e = 2.71828.}

ANSWER:

	Functions		Similarity group
F(1)	(n – 2)!	(n-2)! => O((n-2)!) This a factorial and can implement by a recursive call from 1 to n-2	
F(2)	2^2n	2^2n = (2^2) ^n = 4^n => O(4^n)	Group 1

F(3)	0.002*n^4 + 3*n^2 +1	0.002*n^4 + 3*n^2 +1 = infinity + infinity + constant = infinity + infinity + 0 => Therefore, the time complexity will be O(n^4) as n^4 is the highest	Group 2
F(4)	2^n	2^n => O(2^n)	Group 1
F(5)	e^n	e^n => O(e^n) => e is a constant there for e^n belongs to Group 1	Group 1
F(6)	n*2^n	n*2^n => O(n2^n) => Due to the time complexity of this function is based on ^n. Therefore it belongs to group 1	Group 1
F(7)	1*n^2*n	1*n^2*n = n^2n =>, since 2 is a constant therefore the time complexity of this function, is O(n^2n)	
F(8)	3√ n	3√ n => O(n^1/3)	Group 2
F(9)	3^n	3^n => O(3^n)	Group 1
F(10)	2^log n	2^log n => O(2^log n) the logarithmic function takes less time complexity than other given function	Group 3
F(11)	n^2	n^2 => O(n^2)	Group 2
F(12)	4^log n	4^log n => O(4^log n) the logarithmic function takes less time complexity than other given function	Group 3
F(13)	√log n	√log n = O(log n^1/2)	

<u>Group:</u>

- Group 1: F(2), F(4), F(5), F(6), F(9)
- Group 2: F(3), F(11), F(8)
- Group 3: F(10), F(12)
- Standalone: F(1), F(7), F(13)

<u>Total Comparison:</u>

F(1) = O((n-2)!)
F(7) = O(n^2n)
F(13) = O(log n^1/2): will take less than other functions because ^½ is a constant. Therefore, O(log n^1/2) is equal to O(log n) (in the order of growth)
Group 3 sampling: F(10) = O(2^log n): second smallest in the order of growth in logarithmic function takes less than other given function
Group 1 sampling: F(2) = O(4^n)
Group 2 sampling: F(3) = O(n^4): third smallest

$F(13) < \text{Group3} < \text{Group2} < \text{Group1} < F(7) < F(1)$

<u>Group 1 Comparision:</u>

We have Group 1: F(2), F(4), F(5), F(6), F(9)

$F(2) = O(4^n)$

$F(4) = O(2^n)$

$F(5) = O(e^n) = O(2.71828^n)$

$F(6) = O(n2^n)$

$F(9) = O(3^n)$

Therefore, the order of growth within group 1 should be: $F(4)<F(5)<F(9)<F(2)<F(6)$

<u>Group 2 Comparision:</u>

We have Group 2: F(3), F(8), F(11)

$F(3) = O(n^4)$

$F(8) = O(n^{1/3})$

$F(11) = O(n^2)$

Therefore, the order of growth within group 1 should be: $F(8)<F(11)<F(3)$

<u>Group 3 Comparision:</u>

We have Group 3: F(10), F(12)

$F(10) = O(2^{\log n})$

$F(12) = O(4^{\log n})$

Therefore, the order of growth within group 1 should be: $F(10)<F(12)$

Finally: $F(13) < \text{Group3} < \text{Group2} < \text{Group1} < F(7) < F(1)$

<u>Final Result:</u>

$F(13)<F(10)<F(12)<F(8)<F(11)<F(3)<F(4)<F(5)<F(9)<F(2)<F(6) <= F(7)<F(1)$

Therefore:

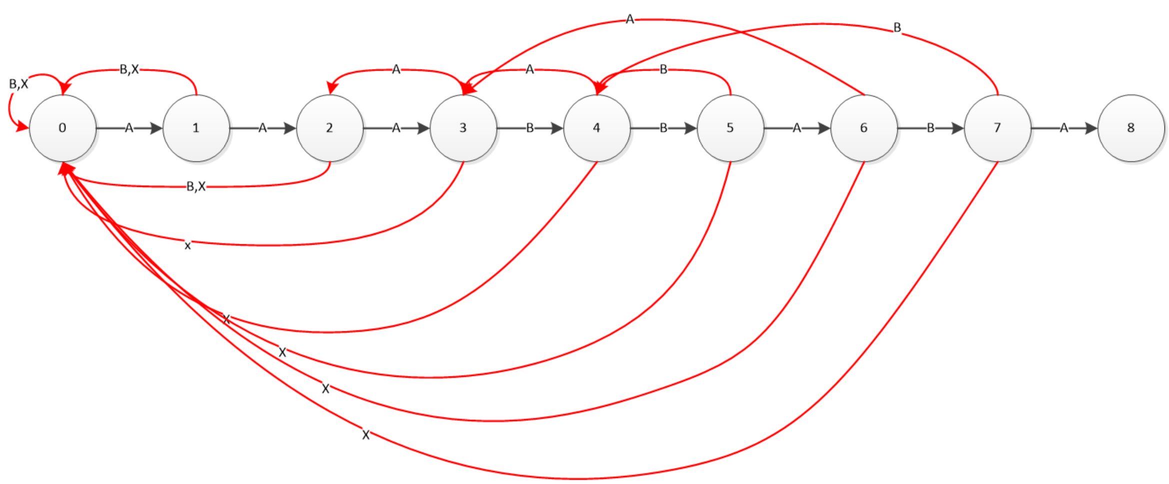
$[\sqrt{\log n}] < [2^{\log n}] < [4^{\log n}] < [3\sqrt{n}] < [n^2] < [0.002*n^4 + 3*n^2 +1] < [2^n] < [e^n] < [3^n] < [2^{2n}] < [n*2^n] = [1*n^2*n] < (n-2)!$

Problem 3[40 points]:
Construct the string-matching automaton for the pattern P = aaabbaba over the alphabet Σ = {a, b, x |x is any letter other than a and b}; and illustrate its operation on the text string T = aaaabbabaaabbaaabbabaab.

3a. Construct the string-matching automation for the pattern P over the alphabet Σ = {a, b, x}in terms of the state transition table (Complete the state transition table)

ANSWER:

	input			P
state	a	b	x	
0	1	0	0	a
1	2	0	0	a
2	3	0	0	a
3	2	4	0	b
4	3	5	0	b
5	6	4	0	a
6	3	7	0	b
7	8	4	0	a
8				



3b. Show the operation on the text string T, computed by the state transition table. Complete the following table, in which T[i] is the letter at the position i of the text string; and State Φ(T[i]) stands for the state transition Φ (s, T[i]) = s'.

text string T = aaaabbabaaabbaaabbabaab.

ANSWER:

Stop when hit step 8, I found the pattern matches the text string at index i=9. I then stop to fill in the table because I have already found the matched string there is no need to keep going

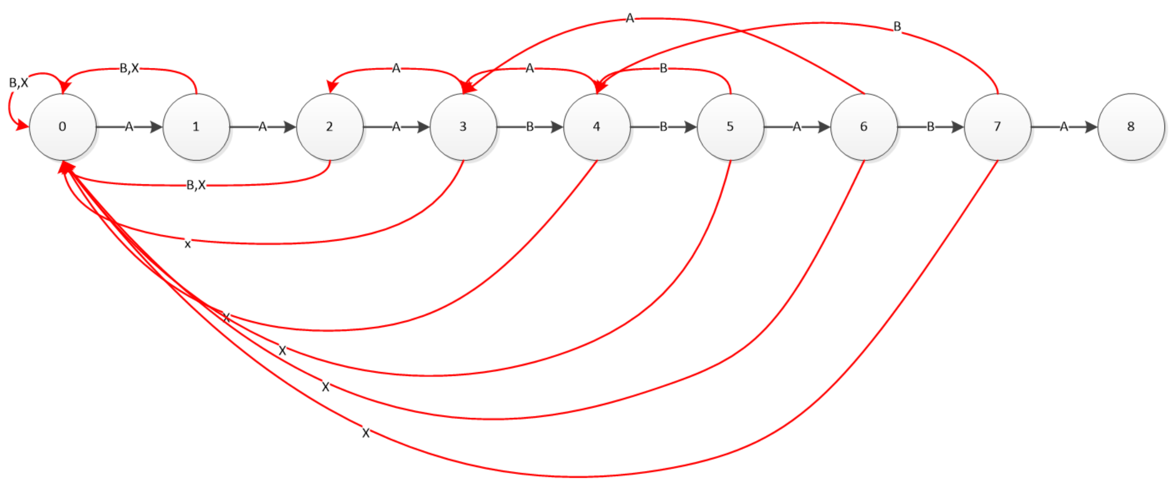
i		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
T[i]		a	a	a	a	b	b	a	b	a	a	a	b	b	a	a	a	b	b	a	b	a	a	b	
State Φ(T[i])	0	1	2	3	3	4	5	6	7	8															

3c. Complete the following sentence.
ANSWER:

The result is text string T = "aaaabbabaaabbaaabbabaab" matches the pattern at shift = 1 (i =2) and shift = 8 (i =9). (Note that shift = i – 1)

3d. Draw a state transition diagram for a string-matching automaton for the pattern P over the alphabet Σ = {a, b, x |x is any letter other than a and b}.
ANSWER:

The state transaction diagram was drawn with Microsoft Visio.



Problem 4[30 points]

Consider the following Algorithm Quicksort:

Algorithm Quicksort(A[p .. r])
//Quicksort(A[0 .. n - 1]) is the initial call for sorting an entire array A.

Input: A subarray A[p .. r] of A[0 .. n-1], defined by its left and right indices p and r.

Output: Subarray A[p .. r] sorted in nondecreasing order

```
{
if (p < r )
        {s ← Partition(A[p .. r]);  //s ← j is a split position
         Quicksort(A[p .. s-1]);    //there is s - p elements
         Quicksort(A[s+1 .. r]);}   //there is r - s elements
}//end of Quicksort()
```

Algorithm Partition(A[p .. r])
//Partitions a subarray by using its first element as a pivot.

Input: A subarray A[p .. r] of A[0 .. n-1], defined by its left and right indices p and r, (p < r).

Output: A partition of A[p .. r], with the split position returned as this function's value

```
{
x ← A[p];       //set x be the leftmost element of A[p .. r].
i ← p; j ← r+1;  //set left and right pointers pointing at p and r+1
repeat
    repeat i ← i+1 until A[i] ≥ x; //move i towards right until …
    repeat j ← j-1 until A[j] ≤ x; //move j towards left until …
    swap(A[i], A[j]);
until i ≥ j;
swap(A[i], A[j]);
swap(A[p], A[j]);
return j;
}
```

In the Algorithm Partition(A[p .. r]), there are three swap() procedures.

4a. When and why the first swap(A[i], A[j]) is needed?

ANSWER:

- The first swap(A[i], A[j]) is needed because this first swap enables i and j to continue to move.

- The move until i and j cross each other or both i and j point at the same place.
- We also need to increment i and decrement j after each swap.

4b. When and why the second swap(A[i], A[j]) is needed?

ANSWER:

- The second swap (A[i], A[j]) is needed to undo the last swap when i>= j.
- This means when i>= j we need to partition the array after exchanging the pivot with A[j]

4c. When and why the third swap(A[p], A[j]) is needed?

ANSWER:

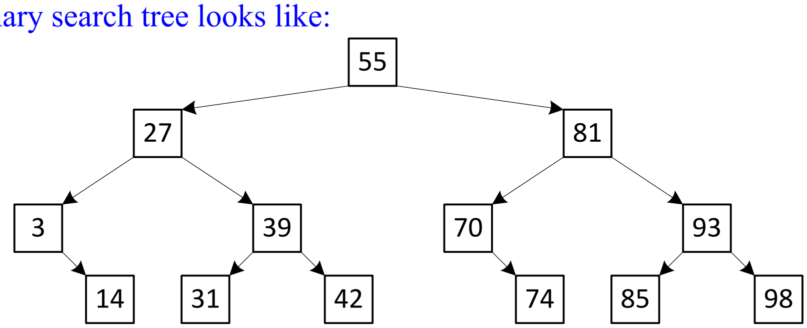
- The third swap is needed to exchange the pivot A[p] with A[j] whenever i>j

Problem 5 [70 points]

Given the following array A[0..15] contains 13 elements.

	3	14	27	31	39	42	55	70	74	81	85	93	98		

It is helpful when answering these questions 5a through 5d to recognize what an equivalent binary search tree looks like:



5a. What is the largest number of key comparisons made by binary search in searching for a key in the following array?

ANSWER:

Both procedures run in O(h) time on a tree of height h = Θ(log n). Θ(log n) in the average case

The number of elements in an array n=13 maximum operation. Thus Cworst(n) = ± $\log_2(n+1) = \log_2(13+1) = 3.907352 = 4$.

5b. List all the keys of this array that will require the largest number of key comparisons when searched for by binary search.

ANSWER:

Value	3	14	27	31	39	42	55	70	74	81	85	93	98
Key	0	1	2	3	4	5	6	7	8	9	10	11	12

According to the tree structure, the lowest level requires the largest key comparison. They are:

Value	14	31	42	74	85	98
Key	1	3	5	8	10	12

5c. Find the average number of key comparisons made by binary search in a successful search in this array. (Assume that each key is searched for with the same probability.)

ANSWER:

The average number of key comparisons made by binary search in a successful search is:

$= 1 * (1/13) + 2 * (2/13) + 3 * (4/13) + 4 * (6/13) = 1/13 + 4/13 + 12/13 + 24/13 = (1+4+12+24) / 13 = 41/13 = 3.2$

5d. Find the average number of key comparisons made by binary search in an unsuccessful search in this array. (Assume that searches for keys in each of the 14 intervals formed by the array's elements are equally likely.)

ANSWER:

There are 3 comparisons at position 6, or 7 (the key is at level 0 and between positions 6 and 7). For the remaining 12 elements, there will be 4 comparisons occurring. The average number of key comparisons made by binary search in an unsuccessful search is:

3*(2/14) + 4*(12/14) = (48+6)/14 = 54/14 = 3.9

5e. Assume that the arrival of the elements is in the order 3, 14, 27, …., 98 of a given array A[0..15]. Rearrange the contents of 13 elements such that array A forms an AVL tree. Show step-by-step in terms of the intermediate resulting arrays.

ANSWER:

The design is drawn using MS Visio. The sourcee is here: Drawing.vsdx

AVL Tree is a self-balancing BST. There is 4 possible rotation: LL Rotation, RR Rotation, LR Rotation, RL Rotation

	3	14	27	31	39	42	55	70	74	81	85	93	98		

Insert 3:

Node	Height
3	0

no rotation

	3														

Insert 14:

Node	Height
3	-1
14	0

no rotation

	3		14												

Insert 27:

Node	Height

3	-2
14	-1
27	0

Rotate at 14, and 14 becomes the root

	14	3	27												

Insert 27:

Node	Height
3	0
14	-1
27	-1
31	0

No rotation

	14	3	27				31								

Insert 39:

Node	Height
3	0
14	-2
27	-2
31	-1
39	0

Rotate at 27

	27	14	31	3			39								

Insert 42:

Node	Height
3	
14	

27	
31	-2
39	-1
42	0

Rotate at 31

	27	14	39	3		31	42								

Insert 55:

Node	Height
3	
14	
27	-1
31	-1
39	-1
42	-1
55	0

No rotation I needed

	27	14	39	3		31	42								55

Insert 70:

Node	Height
3	
14	
27	
31	
39	
42	-2
55	-1
70	0

Rotate at 42

	27	14	39	3		31	55							42	70

Insert 74:

Node	Height
3	
14	
27	
31	
39	-2
42	-1
55	
70	-1
74	0

Rotate at 39

	27	14	55	3		39	70					31	42		74

Insert 81:

Node	Height
3	
14	
27	
31	
39	
42	
55	
70	-2
74	-1
81	0

Rotate at 70

	27	14	55	3		39	74					31	42	70	81

Insert 85:

Node	Height
3	
14	
27	-2
31	
39	
42	
55	-1
70	0
74	-1
81	-1
85	0

Rotate at 27

	55	27	74	14	39	70	81	3		31	42				85

Insert 93:

Node	Height
3	
14	
27	
31	
39	
42	
55	
70	
74	
81	-2

85	-1
93	0

Rotate at 81

	55	27	74	14	39	70	85	3		31	42			81	93

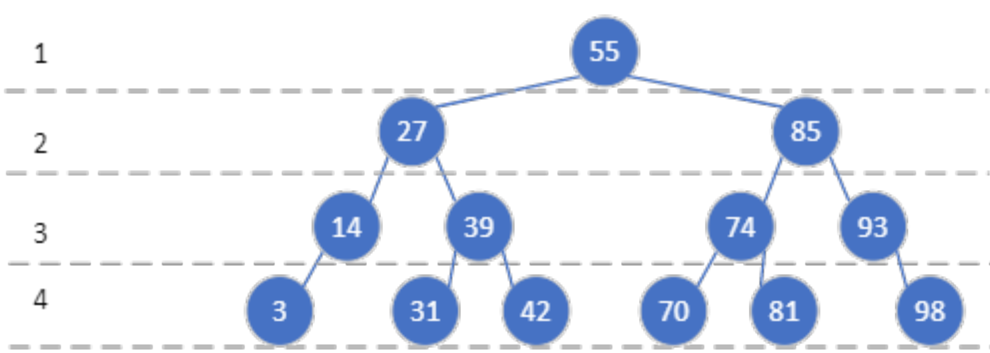
Insert 98:

Node	Height
3	
14	
27	
31	
39	
42	
55	
70	
74	-2
81	
85	-1
93	-1
98	0

Rotate at 74

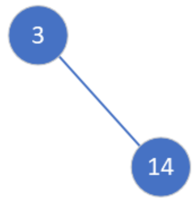
	55	27	85	14	39	74	93	3		31	42	70	81		98

Result:

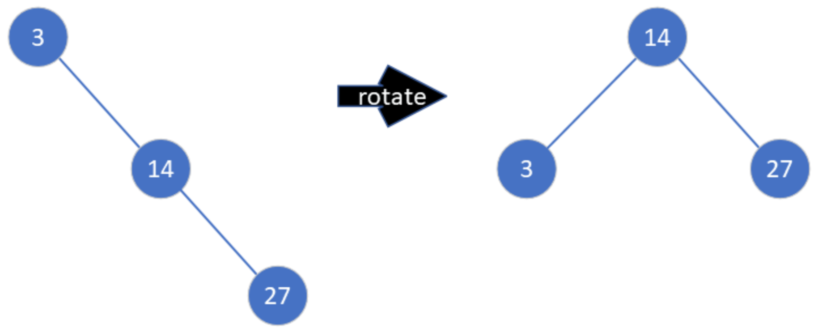


Step 1:Insert 3, 3 now is the root

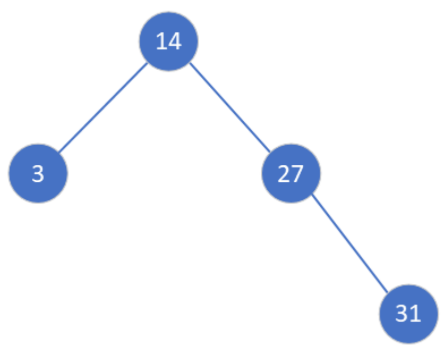
Step 2:Insert 14, 14 is bigger than 3, 14 go to the right of the tree

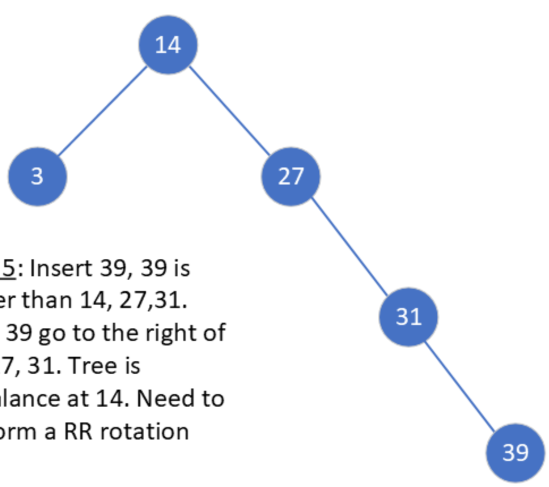


Step 3: Insert 27, 27 is bigger than 3 and 14, 27 go to the right of 14. Now this tree is unbalance and we need to perform a RR rotation

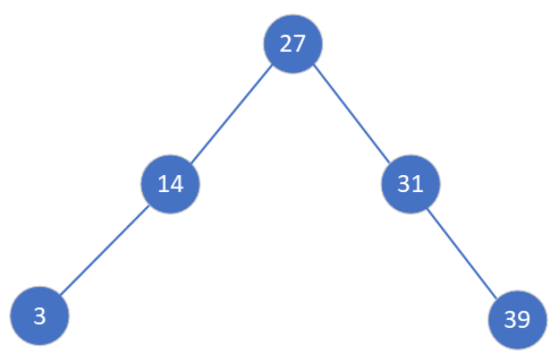


Step 4: Insert 31, 31 is bigger than 14, 27. Thus 31 go to the right of 14, 27. Tree is still balance.

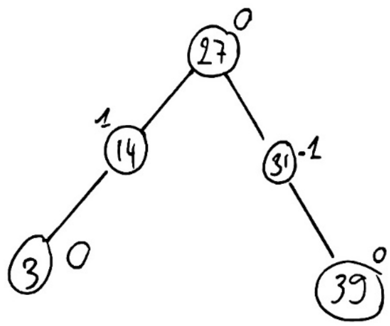




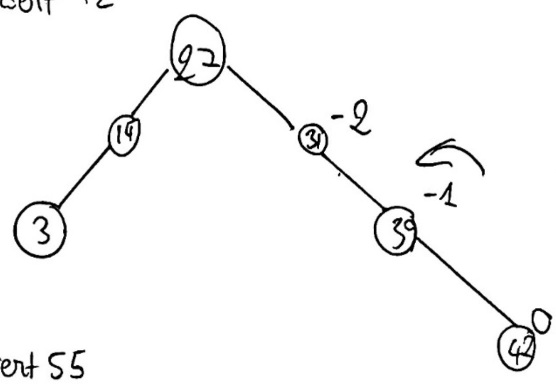
Step 5: Insert 39, 39 is bigger than 14, 27,31. Thus 39 go to the right of 14, 27, 31. Tree is unbalance at 14. Need to perform a RR rotation



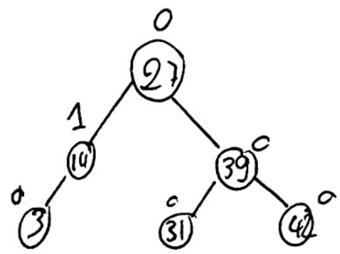
$B = L - R$



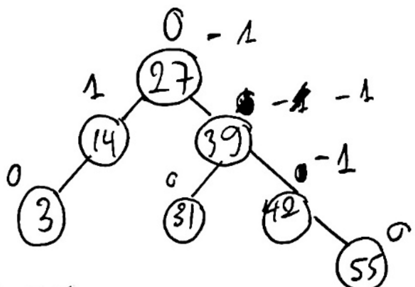
insert 42



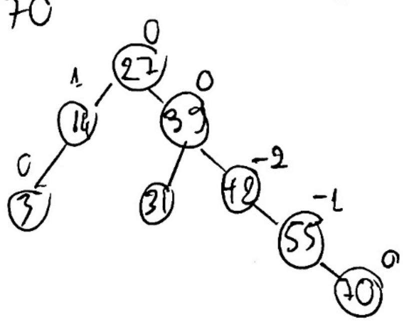
rotate at 31 $\rightarrow$



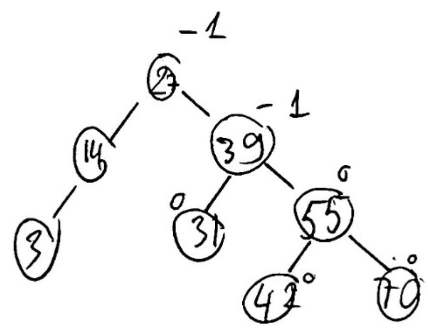
insert 55



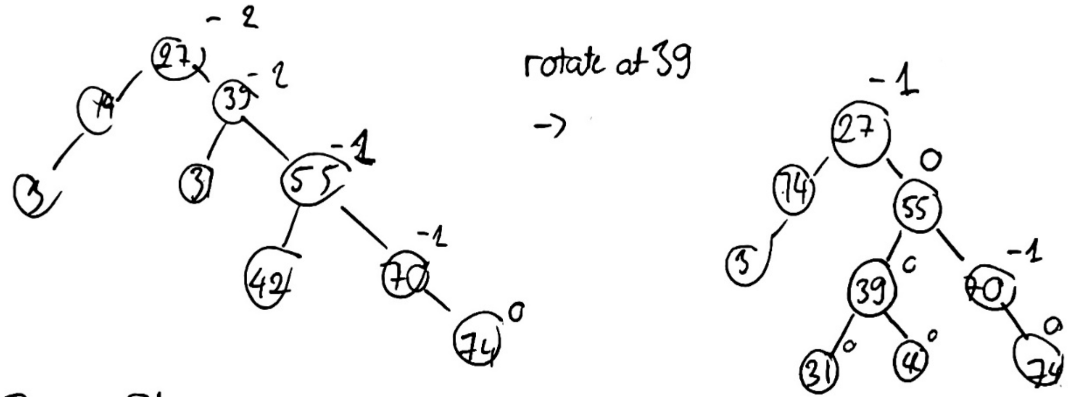
insert 70



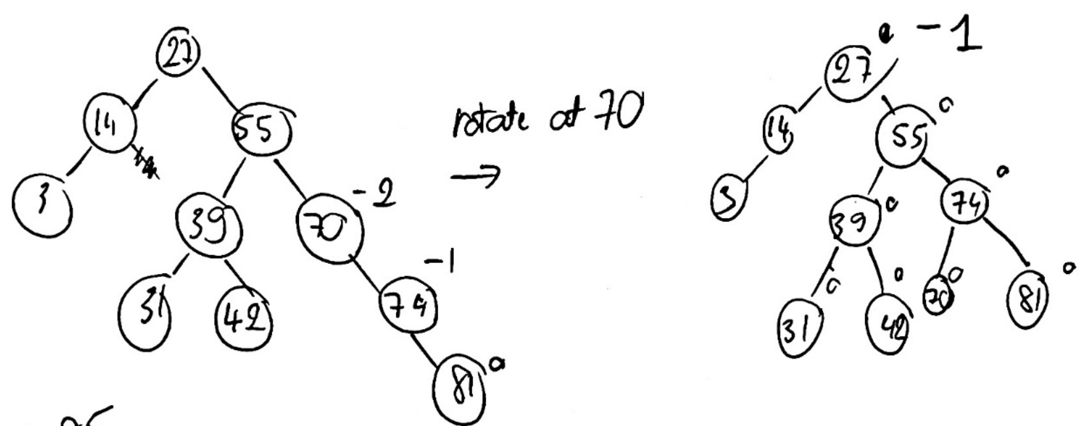
rotate at 42 $\rightarrow$



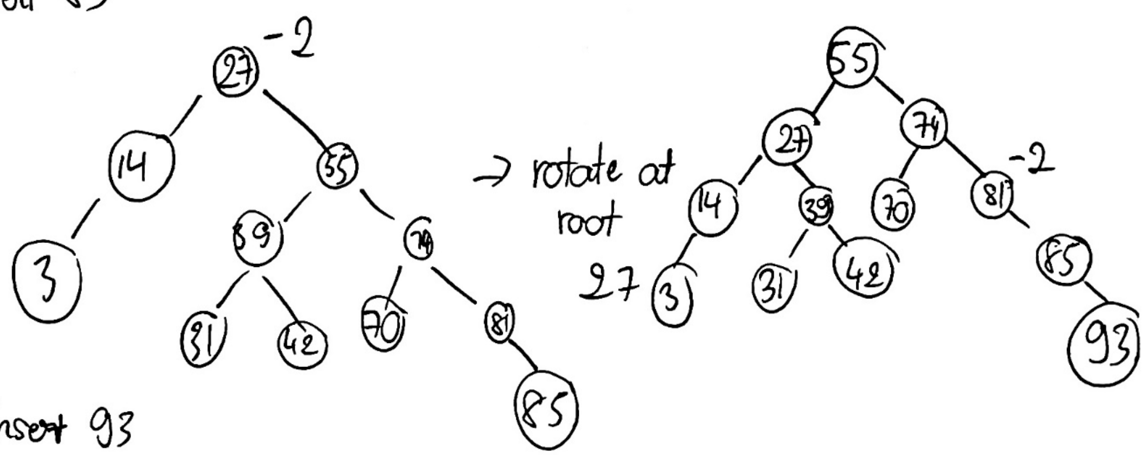
Insert 74:



Insert 81

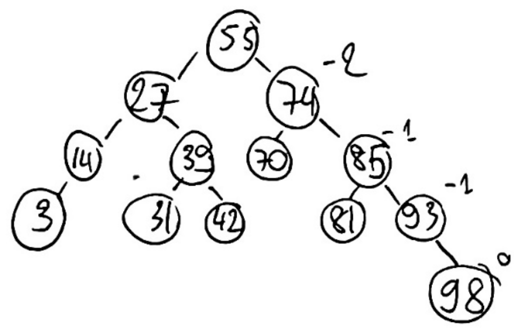


Insert 85

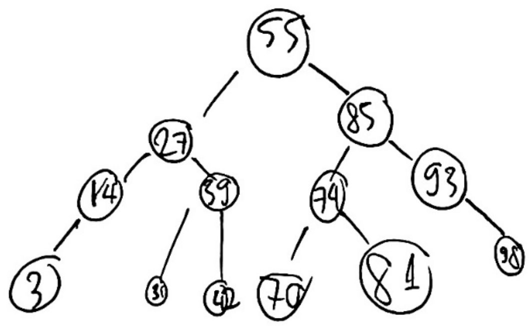


Insert 93

After insert at 93 rotate at 81.



Insert 98



5f. What is the largest number of key comparisons in searching for a key in array A which has an AVL tree?

ANSWER:

Both procedures run in O(h) time on a tree of height h = Θ(log n). Θ(log n) in the average case. The number of elements in an array n=13 maximum operation. Thus largest number of key comparision = $\log_2(n) = \log_2(13) = 3.7 = 4$

5g. List all the keys of this array that will require the largest number of key comparisons when searched for by binary search.

ANSWER:

Key	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Value	55	27	85	14	39	74	93	3		31	42	70	81		98
	55	27	85	14	39	74	93	3		31	42	70	81		98

According to the tree structure, the lowest level requires the largest key comparison. They are:

Value	7	9	10	11	12	14
Key	3	31	42	70	81	98

Problem 6 [20 points]

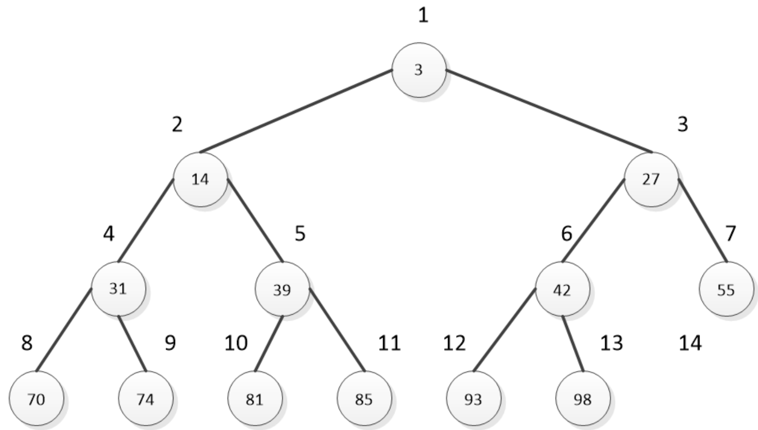
Given the following array A[0..15] contains 13 elements.

	3	14	27	31	39	42	55	70	74	81	85	93	98		

6a. Use Max Heapify(A, i), 0< i to maintain given array A[0..15] a max-heap.

ANSWER:

This is the heap structure before apply the MaxHeapify



This structure violate the definition of maxheap. Therefore, we need to reconstruct

	3	14	27	31	39	42	55	70	74	81	85	93	98		

<u>1^{st} run:</u>

Run Heapify from └ lengthA[ ]/2 ┘ to 1, to determine their children node, A[2i] or A[2i+1]. Therefore, we start at index i= 12/2 = 6 and 42, 93 < 98. Therefore swap 98 and 42.

	3	14	27	31	39	98	55	70	74	81	85	93	42		

Run Heapify again since i-- now i=5 and 39,81 < 85. Therefore swap 39 and 85

	3	14	27	31	85	98	55	70	74	81	39	93	42		

Run Heapify again since i-- now i=4 and 31,70 < 74. Therefore swap 31 and 74

	3	14	27	74	85	98	55	70	31	81	39	93	42		

Run Heapify again since i-- now i=3 and 27,55 < 98. Therefore swap 27 and 98

	3	14	98	74	85	27	55	70	31	81	39	93	42		

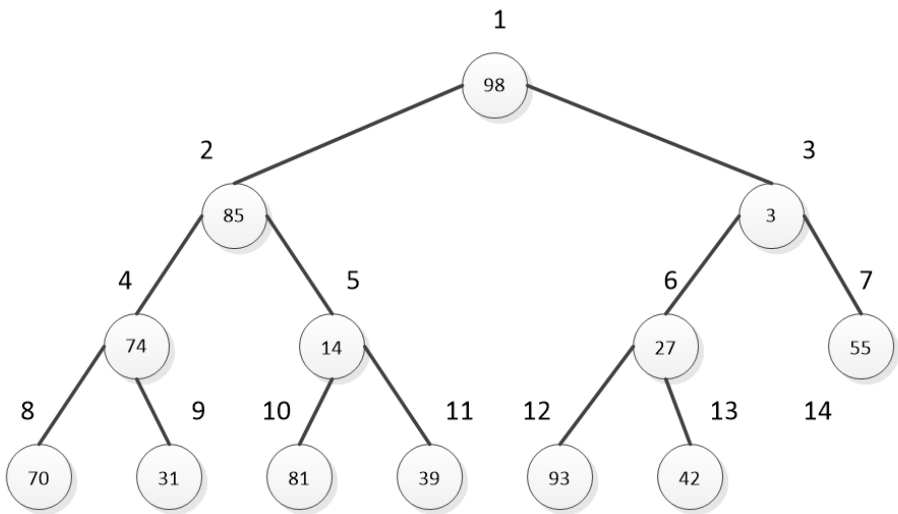
Run Heapify again since i-- now i=2 and 14,74 < 85. Therefore swap 14 and 85

	3	85	98	74	14	27	55	70	31	81	39	93	42		

Run Heapify again since i-- now i=1 and 3,85 < 98. Therefore swap 3 and 98

	98	85	3	74	14	27	55	70	31	81	39	93	42		

This is the heap structure after 1st round



At this point we run function to validate if this is a maxheap. And it is not therefore, we recursive run the heapify again.

	98	85	3	74	14	27	55	70	31	81	39	93	42		

<u>2nd run:</u>

Run Heapify from $\lfloor$ lengthA[ ]/2 $\rfloor$ to 1, to determine their children node, A[2i] or A[2i+1]. Therefore, we start at index i= 12/2 = 6 and 27, 42 < 93. Therefore swap 93 and 27.

	98	85	3	74	14	93	55	70	31	81	39	27	42		

Run Heapify again since i-- now i=5 and 14,39 < 81. Therefore swap 14 and 81

	98	85	3	74	81	93	55	70	31	14	39	27	42		

Run Heapify again since i-- now i=4 and 70,31 < 74. DO NOTHING

	98	85	3	74	81	93	55	70	31	14	39	27	42		

Run Heapify again since i-- now i=3 and 3,55 < 93. Swap 93 and 3

	98	85	93	74	81	3	55	70	31	14	39	27	42		

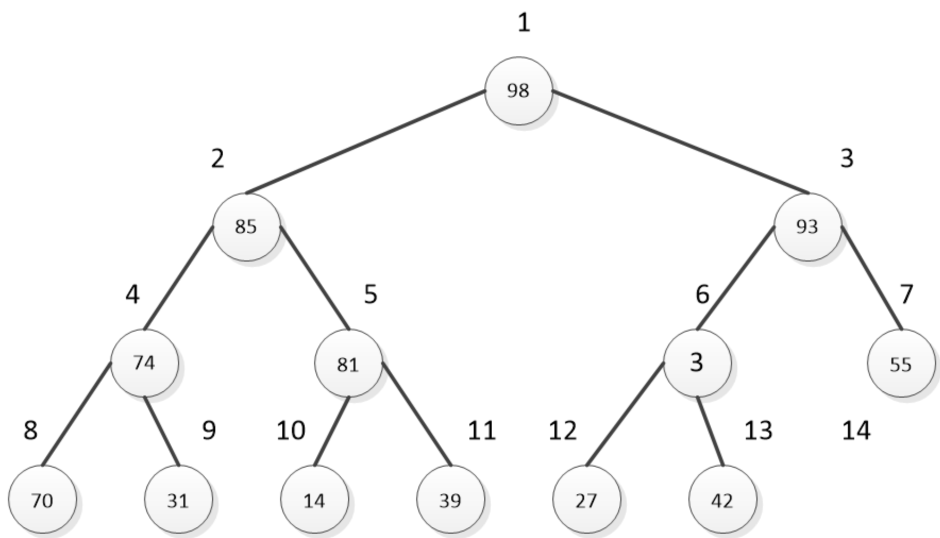
Run Heapify again since i-- now i=2 and 74,81 < 85. Do nothing

	98	85	93	74	81	3	55	70	31	14	39	27	42		

Run Heapify again since i-- now i=1 and 98 is the biggest. Do nothing

	98	85	93	74	81	3	55	70	31	14	39	27	42		

At this point, we runa function to validate if this is a maxheap. And it is not therefore, we recursive run the heapify again.



3rd run:

Start with i= 6 and 3, 27 < 42. Swap 3 and 42.

	98	85	93	74	81	42	55	70	31	14	39	27	3		

After that when i=5,4,3,2,1 do nothing (no swap occurs). Therefore, the final result in the heap is:

	98	85	93	74	81	42	55	70	31	14	39	27	3		

6b. Using the resultant max-heap array obtained from 6a, apply the heapsort algorithm to obtain a sorted array A in descending order. Show step-by-step in terms of the intermediate resulting arrays.

ANSWER:

We have the current max heap

	98	85	93	74	81	42	55	70	31	14	39	27	3		

I draw a tree structure on my whiteboard to help me visualize and work easier. For every time the max heap swap and restructure I re-draw the tree.

Swap the first and last node

	3	85	93	74	81	42	55	70	31	14	39	27	98		

And take 98 out of the maxheap and put it in the sorted array. Please note the one in red is marked as not existing in the maxheap but exists in the sorted array. The max heap now looks like 3, 85, 93, 74, 81, 42, 55, 70, 14, 39, 27 (1).

Restructure the max heap

	93	85	55	74	81	42	3	70	31	14	39	27	98		

Swap the first and last node, similar to (1)

	27	85	55	74	81	42	3	70	31	14	39	93	98		

Restructure the max heap

	85	81	55	74	39	42	3	70	31	14	27	93	98		

Swap the first and last node, similar to (1)

	27	81	55	74	39	42	3	70	31	14	85	93	98		

Restructure the max heap

	81	74	55	70	39	42	3	27	31	14	85	93	98		

Swap the first and last node, similar to (1)

	14	74	55	70	39	42	3	27	31	81	85	93	98		

Restructure the max heap

	74	70	55	31	39	42	3	27	14	81	85	93	98		

Swap the first and last node

	14	70	55	31	39	42	3	27	74	81	85	93	98		

Restructure the max heap, similar to (1)

	70	39	55	31	14	42	3	27	74	81	85	93	98		

Swap

	27	39	55	31	14	42	3	70	74	81	85	93	98		

Restructure the max heap, similar to (1)

	55	39	42	31	14	27	3	70	74	81	85	93	98		

Swap

	3	39	42	31	14	27	55	70	74	81	85	93	98		

Restructure the max heap, similar to (1)

	42	39	27	31	14	3	55	70	74	81	85	93	98		

Swap

	3	39	27	31	14	42	55	70	74	81	85	93	98		

Restructure the max heap, similar to (1)

	39	31	27	3	14	42	55	70	74	81	85	93	98		

Swap

	14	31	27	3	39	42	55	70	74	81	85	93	98		

Restructure the max heap, similar to (1)

	31	14	27	3	39	42	55	70	74	81	85	93	98		

Swap

	3	14	27	31	39	42	55	70	74	81	85	93	98		

Restructure the max heap, similar to (1)

	27	14	3	31	39	42	55	70	74	81	85	93	98		

Swap

	3	14	27	31	39	42	55	70	74	81	85	93	98		

Restructure the max heap, similar to (1)

	14	3	27	31	39	42	55	70	74	81	85	93	98		

Swap

	3	14	27	31	39	42	55	70	74	81	85	93	98		

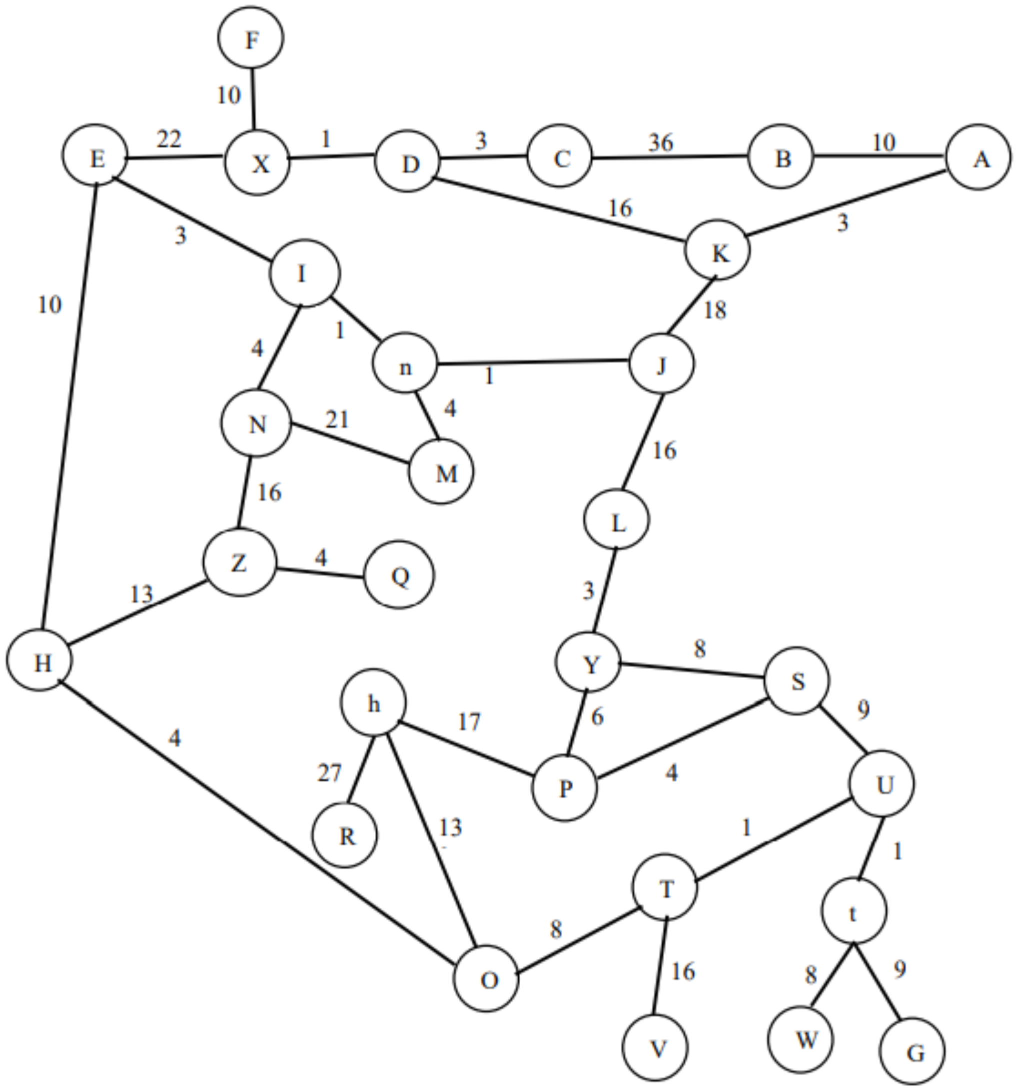
Terminate now we got an ascending order; reversing the array to get a descending order.

The result is

	98	93	85	81	74	70	55	42	39	31	27	14	3		

Problem 7 [50 points]

Given a weighted graph G, which is as follows:



7a. Construct

(i) a weighted adjacency list and

(ii) a weighted adjacency matrix

ANSWER:

Vertex List

0	F
1	E
2	X
3	D
4	C
5	B
6	A
7	I
8	K
9	N
10	n
11	J
12	M
13	L
14	Z
15	Q
16	H
17	h
18	Y
19	S
20	R
21	P
22	U
23	O
24	T
25	t
26	V
27	W
28	G

	F	E	X	D	C	B	A	I	K	N	n	J	M	L	Z	Q	H	Y	S	h	R	P	U	O	T	t	V	W	G
F	∞	∞	10	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
E	∞	∞	22	∞	∞	∞	∞	3	∞	∞	∞	∞	∞	∞	∞	∞	10	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
X	10	22	∞	1	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
D	∞	∞	1	∞	3	∞	∞	∞	16	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
C	∞	∞	∞	3	∞	36	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
B	∞	∞	∞	∞	36	∞	10	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
A	∞	∞	∞	∞	∞	10	∞	∞	3	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
I	∞	3	∞	∞	∞	∞	∞	∞	∞	4	1	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
K	∞	∞	∞	16	∞	∞	3	∞	∞	∞	∞	18	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
N	∞	∞	∞	∞	∞	∞	∞	4	∞	∞	∞	∞	21	∞	16	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
n	∞	∞	∞	∞	∞	∞	∞	1	∞	∞	∞	1	4	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
J	∞	∞	∞	∞	∞	∞	∞	∞	18	∞	1	∞	∞	16	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
M	∞	∞	∞	∞	∞	∞	∞	∞	∞	21	4	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
L	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	16	∞	∞	∞	∞	∞	3	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
Z	∞	∞	∞	∞	∞	∞	∞	∞	∞	16	∞	∞	∞	∞	∞	4	13	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
Q	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	4	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
H	∞	10	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	13	∞	∞	∞	∞	∞	∞	∞	∞	4	∞	∞	∞	∞	∞
Y	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	3	∞	∞	∞	∞	8	∞	∞	6	∞	∞	∞	∞	∞	∞	∞
S	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	8	∞	∞	∞	4	9	∞	∞	∞	∞	∞	∞
h	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	27	17	∞	13	∞	∞	∞	∞	∞
R	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	27	∞	∞	∞	∞	∞	∞	∞	∞	∞
P	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	6	4	17	∞	∞	∞	∞	∞	∞	∞	∞	∞
U	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	9	∞	∞	∞	∞	∞	1	1	∞	∞	∞
O	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	4	∞	∞	13	∞	∞	∞	∞	8	∞	∞	∞	∞
T	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	1	8	∞	∞	16	∞	∞
t	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	1	∞	∞	∞	∞	8	9
V	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	16	∞	∞	∞	∞
W	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	8	∞	∞	∞
G	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	9	∞	∞	∞
	F	E	X	D	C	B	A	I	K	N	n	J	M	L	Z	Q	H	Y	S	h	R	P	U	O	T	t	V	W	G

The *weight adjacency-matrix*

I used Microsoft Excel to create the table. The original source can be found at:

Final.xlsx

F	→	X, 10				
E	→	X, 22	→	I, 3	→	H, 10
X	→	F, 10	→	E, 22	→	D, 1
D	→	X, 1	→	C, 3	→	K, 16
C	→	D, 3	→	B, 36		
B	→	C, 36	→	A, 10		
A	→	B, 10	→	K, 3		
I	→	E, 3	→	N, 4	→	n, 1
K	→	D, 16	→	A, 3	→	J, 18
N	→	I, 4	→	M, 21	→	Z, 16
n	→	I, 1	→	J, 1	→	M, 4
J	→	K, 18	→	n, 1	→	L, 16
M	→	N, 21	→	n, 4		
L	→	J, 16	→	Y, 3	→	
Z	→	N, 16	→	Q, 4	→	H, 13
Q	→	Z, 4				
H	→	E, 10	→	Z, 13	→	O, 4
Y	→	L, 3	→	S, 8	→	P, 6
S	→	Y, 8	→	P, 4	→	U, 9
h	→	R, 27	→	P, 17	→	O, 13
R	→	h, 27				
P	→	Y, 6	→	S, 4	→	h, 17
U	→	S, 9	→	T, 1	→	t, 1
O	→	H, 4	→	h, 13	→	T, 8
T	→	U, 1	→	O, 8	→	V, 16
t	→	U, 1	→	W, 8	→	G, 9
V	→	T, 16				
W	→	t, 8				
G	→	t, 9				

The *weight adjacency-list*
I used Microsoft Excel to create the table. The original source can be found at:
Final.xlsx

Traversing the given graph, based on its weighted adjacency list representation obtained in problem 7a(i), construct its depth-first search tree forest starting from vertex A. In your obtained DFS tree forest, show the tree edges (indicated as solid lines) and back edges (indicated as dotted lines) for your trees. Traversal's stack contains symbols (such as Vi, j, the first subscript number indicates the order in which a vertex V was first visited, say at i, (pushed onto the stack, V), where $0 < i \leq n$; the second one indicates the order in which it became a dead-end, say at j (popped off the stack V), where $0 < j < n$. n is the total number of vertices for the given graph. For simplicity's sake, please use two time-stamps: one is $0 < i \leq n$, the order for pushing a vertex onto the stack counting from 1 through n. The other one is $0 < j \leq n$, the order for popping off a vertex from the stack counting from 1 through n. For this problem, you need to answer 7b through 7e, which are as follows:

7b. Show the traversal's stack with time-stamp, and what are the orderings of vertices yielded by the DFS?

ANSWER:

			$W_{28,10}$	$G_{29,11}$	
		$Q_{24,7}$	$t_{27,12}$	$t_{27,12}$	
		$Z_{23,8}$	$U_{26,13}$	$U_{26,13}$	$V_{30,14}$
		$H_{22,9}$	$T_{25,15}$	$T_{25,15}$	$T_{25,15}$
	$R_{20,6}$	$O_{21,16}$	$O_{21,16}$	$O_{21,16}$	$O_{21,16}$
$A_{13,1}$	$h_{19,17}$	$h_{19,17}$	$h_{19,17}$	$h_{19,17}$	$h_{19,17}$
$B_{12,2}$	$P_{18,18}$	$P_{18,18}$	$P_{18,18}$	$P_{18,18}$	$P_{18,18}$
$C_{11,3}$	$S_{17,19}$	$S_{17,19}$	$S_{17,19}$	$S_{17,19}$	$S_{17,19}$
$D_{10,4}$	$Y_{16,20}$	$Y_{16,20}$	$Y_{16,20}$	$Y_{16,20}$	$Y_{16,20}$
$K_{9,5}$	$L_{15,21}$	$L_{15,21}$	$L_{15,21}$	$L_{15,21}$	$L_{15,21}$
$J_{8,22}$	$J_{8,22}$	$J_{8,22}$	$J_{8,22}$	$J_{8,22}$	$J_{8,22}$
$n_{7,23}$	$n_{7,23}$	$n_{7,23}$	$n_{7,23}$	$n_{7,23}$	$n_{7,23}$
$M_{6,24}$	$M_{6,24}$	$M_{6,24}$	$M_{6,24}$	$M_{6,24}$	$M_{6,24}$
$N_{5,25}$	$N_{5,25}$	$N_{5,25}$	$N_{5,25}$	$N_{5,25}$	$N_{5,25}$
$I_{4,26}$	$I_{4,26}$	$I_{4,26}$	$I_{4,26}$	$I_{4,26}$	$I_{4,26}$
$E_{3,27}$	$E_{3,27}$	$E_{3,27}$	$E_{3,27}$	$E_{3,27}$	$E_{3,27}$
$X_{2,28}$	$X_{2,28}$	$X_{2,28}$	$X_{2,28}$	$X_{2,28}$	$X_{2,28}$
$F_{1,29}$	$F_{1,29}$	$F_{1,29}$	$F_{1,29}$	$F_{1,29}$	$F_{1,29}$

Ordering of verticles yield by DFS:
A, B, C, D, K, R, Q, Z, H, W, G, t, U, V, T, O, h, P, S, Y, L, J, n, M, N, I, E, X, F

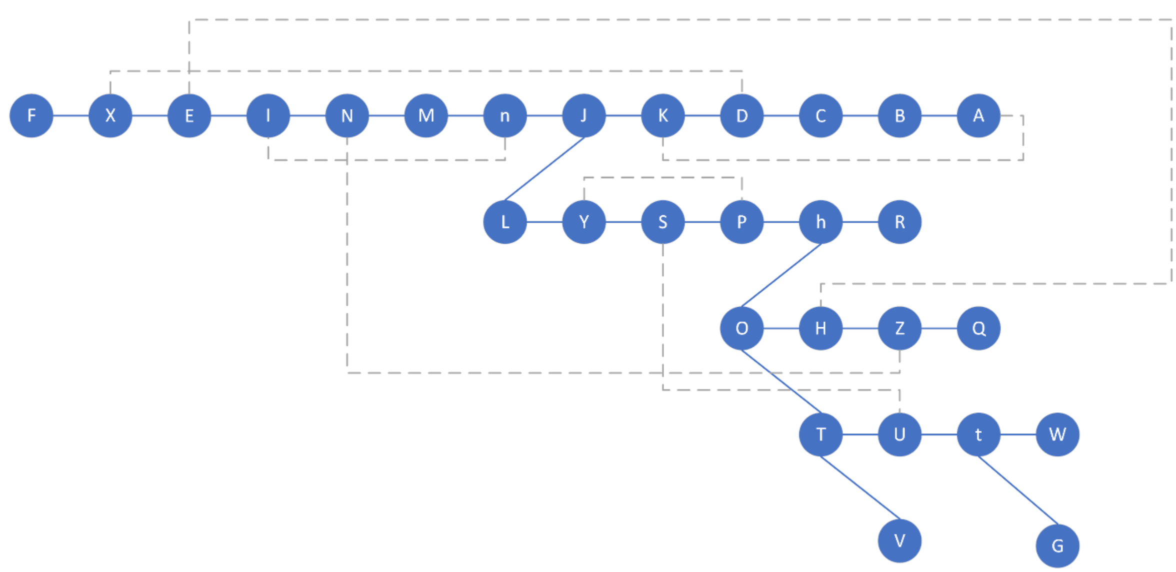
7c. Construct the corresponding depth-first search (DFS) tree forest, with indications of tree edges and back edges.

ANSWER:

I am using Microsoft Visio to construct the DFS tree. Link is here visio-link

Based on the traversal's stack with the time-stamp table, I constructed depth-first search (DFS) tree forest using the following steps:

- At the top level, I have F, X, E, I, N, M, n, J, K, D, C, B, and A are connected by edges
- Since we pop A, B, C, D, K. I connect J to L by an edge
- The next level will be L, Y, S, P, h, and R are connected by edges
- Since we pop R. I connect h to O
- The next level will be O, H, Z, and Q are connected by edges
- Since we will pop Q, Z, H. I connect O to T
- The next level will be T, U, t, and W are connected by edges
- Since we will pop W. I connect G to t, and T to V
- That is finish all the edges of the DFS tree forest
- Now I look at the weight adjacency list to construct the back edges
- Please note **tree edges** are indicated as blue solid lines and **back edges** are indicated as dotted lines



7d. What is the graph called? Is this graph acyclic? Does the graph have articulation points? What is the topological sort ordering for the graph?

ANSWER:

- The graph is called the depth-first forest
- This graph is not acyclic since there are back-edges from some vertex to its ancestor (e.g. X is connected to D via back-edge, E is connected to H via back-edge…)
- The topological sort ordering for the graph is the reverse of pop-off ordering: {F, X, E, I, N, M, n, J, L, Y, S, P, h, O, T, V, U, t, G, W, H, Z, Q, R, K, D, C, B, A}
- Yes, the graph has articulation points.

7e. What are the time efficiency and space efficiency of the DFS?

ANSWER:

- Time efficiency is a linear-time procedure which means running time increases at most linearly with the size of the input.
- Space efficiency is the total space that needs to store all the data structure below in computer memory:
    - a data structure to store graph G (vertex and edges)
    - a stack data structure (list implementation) to store vertex V of graph G. Length of the stack should be equal to the total vertex in Graph G
    - a global variable count to use for timestamp

Problem 8 [40 points]

Traversing the graph given in Problem 7, based on its weighted adjacency list representation obtained in Problem 7a(i), construct its breath-first search (BFS) tree forest starting from vertex A. For this, you need to use a queue (note the difference from DFS) to trace the operation of breadth-first search, indicating the order in which the vertices {…, V', V", …} were visited. i.e., the order of the operation of adding several vertices to, or removing a vertex from the queue $\{V_i'',\ldots, V'_{i+1}, V'_{i+2}, \ldots\}$. The order in which vertices are added to the queue (i.e., enqueue operation) is the same order in which they are removed from it (i.e., dequeue operation). Indicate the tree edges (indicated as solid lines) and cross-edges (indicated as dotted lines) for your trees. For this problem, you need to answer 8a through 7e, which are as follows:

8a. Show the traversal's queue with a time-stamp indicating the order in which the vertices were visited, and what is the ordering of vertices yielded by the BFS?

ANSWER:

FIFO QUEUE	VISITED NODE	ORDER OF VERTICLES
∞		
A Enqueue : A Dequeue:	A'→B→K	
B K Enqueue : B, K Dequeue: A	A''→ B →K B' → C →A K' → D →A →J	A
K C Enqueue : C Dequeue: B	B'' → C → A K' → D → A →J C' → D → B	A B
C D J Enqueue : D, J Dequeue: K	K'' → D → A →J C' → D → B D' → X → C → K J' → K → N → L	A B K
D J X n L Enqueue : X, N, L Dequeue: C	C'' → D → B D' → X → C → K J' → K → N → L X'→ F → E → D n'→ I → J → M L'→ J → Y	A B K C
J X n L Enqueue : Dequeue: D	D'' → X → C → K J' → K → N → L X'→ F → E → D n'→ I → J → M	A B K C D

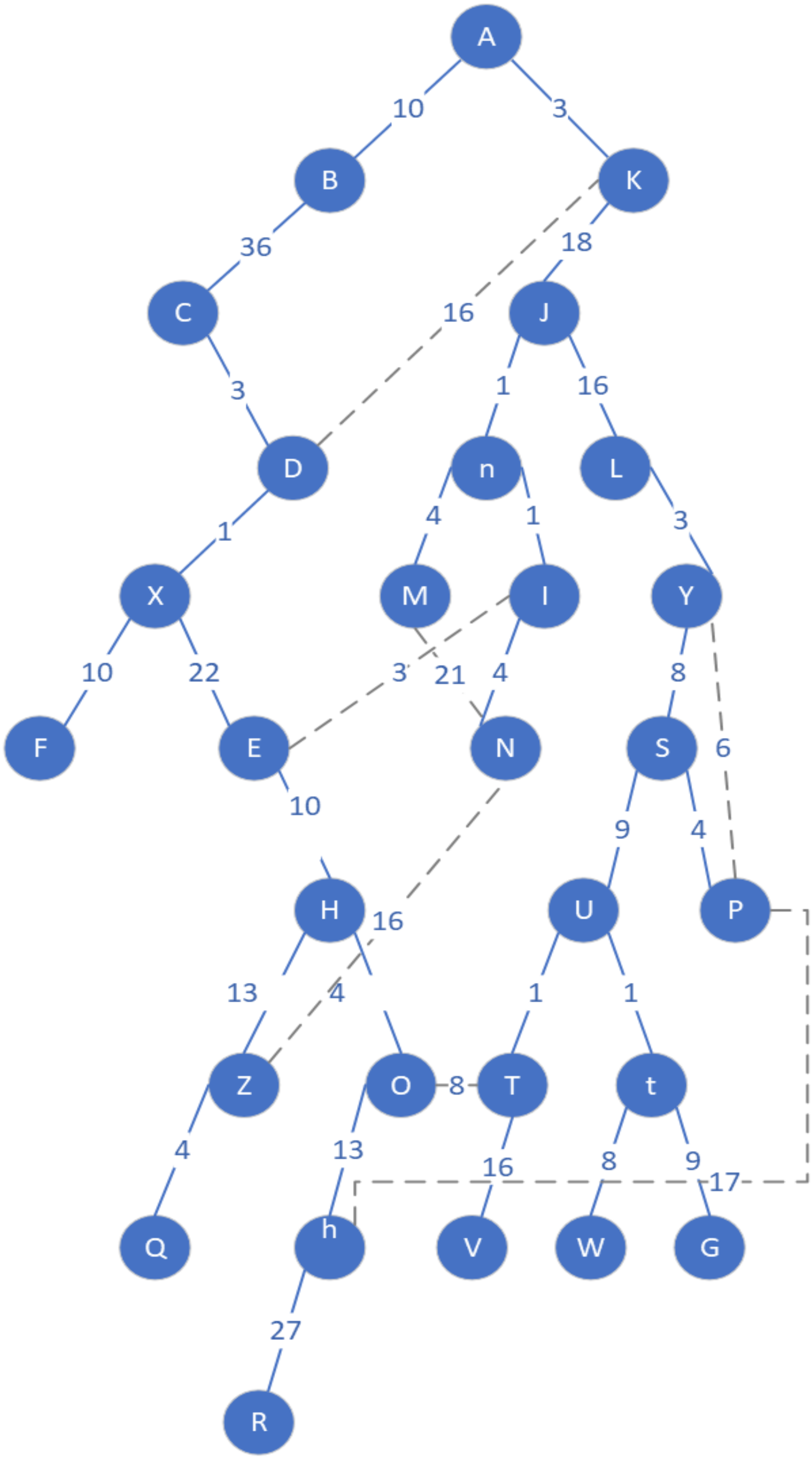
	L’→ J → Y	
X n L Enqueue : Dequeue: J	J’’ → K → N → L X’→ F → E → D n’→ I → J → M L’→ J → Y	A B K C D J
n L F E Enqueue : F E Dequeue: X	X’’→ F → E → D n’→ I → J → M L’→ J → Y F’ → X E’ → X→ I → H	A B K C D J X
n L F E I M Enqueue : I, M Dequeue: n	n’’→ I → J → M L’→ J → Y F’ → X E’ → X→ I → H I’ → E → N → n M’ → N→ n	A B K C D J X n
F E I M Y Enqueue : Y Dequeue: L	L’’→ J → Y F’ → X E’ → X→ I → H I’ → E → N → n M’ → N→ n Y’ → L→ S → P	A B K C D J X n L
E I M Y Enqueue : Dequeue: F	F’’ → X E’ → X→ I → H I’ → E → N → n M’ → N→ n Y’ → L→ S → P	A B K C D J X n L F
I M Y H Enqueue : H Dequeue: E	E’ → X→ I → H I’ → E → N → n M’ → N→ n Y’ → L→ S → P H’ → E→ Z → O	A B K C D J X n L F E
M Y H N Enqueue : N Dequeue: I	I’’ → E → N → n M’ → N→ n Y’ → L→ S → P H’ → E→ Z → O N’ → I→ M → Z	A B K C D J X n L F E I
Y H N Enqueue : Dequeue: M	M’’ → N→ n Y’ → L→ S → P H’ → E→ Z → O N’ → I→ M → Z	A B K C D J X n L F E I M

H N S P Enqueue : S, P Dequeue: Y	Y’’ → L→ S → P H’ → E→ Z → O N’ → I→ M → Z S’ → Y→ P → U P’ → Y→ S → H	A B K C D J X n L F E I M Y
N S P Z O Enqueue : Z, O Dequeue: H	H’’ → E→ Z → O N’ → I→ M → Z S’ → Y→ P → U P’ → Y→ S → H Z’ → N→ Q → H O’→ H→ h → T	A B K C D J X n L F E I M Y H
S P Z O Enqueue : Dequeue: N	N’’ → I→ M → Z S’ → Y→ P → U P’ → Y→ S → H Z’ → N→ Q → H O’→ H→ h → T	A B K C D J X n L F E I M Y H N
P Z O U Enqueue : U Dequeue: S	S’’ → Y→ P → U P’ → Y→ S → H Z’ → N→ Q → H O’→ H→ h → T U’ → S→ T → t	A B K C D J X n L F E I M Y H N S
Z O U Enqueue : Dequeue: P	P’’ → Y→ S → H Z’ → N→ Q → H O’→ H→ h → T U’ → S→ T → t	A B K C D J X n L F E I M Y H N S P
O U Q Enqueue : Q Dequeue: Z	Z’’ → N→ Q → H O’→ H→ h → T U’ → S→ T → t Q’→ Z	A B K C D J X n L F E I M Y H N S P Z
U Q h T Enqueue : h, T Dequeue: O	O’’→ H→ h → T U’ → S→ T → t Q’→ Z h→ R→ P → O T→ U→ O → V	A B K C D J X n L F E I M Y H N S P Z O
Q h T t Enqueue : t Dequeue: U	U’’ → S→ T → t Q’→ Z h’→ R→ P → O T’→ U→ O → V t’→ U→ W → G	A B K C D J X n L F E I M Y H N S P Z O U
h T t Enqueue : Dequeue: Q	Q’’→ Z h’→ R→ P → O T’→ U→ O → V t’→ U→ W → G	A B K C D J X n L F E I M Y H N S P Z O U Q

T t R Enqueue : R Dequeue: h	h’’→ R→ P → O T’→ U→ O → V t’→ U→ W → G R’→ h	A B K C D J X n L F E I M Y H N S P Z O U Q h
t R V Enqueue : V Dequeue: T	T’’→ U→ O → V t’→ U→ W → G R’→ h V’→ T	A B K C D J X n L F E I M Y H N S P Z O U Q h T
t R V Enqueue : V Dequeue: T	T’’→ U→ O → V t’→ U→ W → G R’→ h V’→ T	A B K C D J X n L F E I M Y H N S P Z O U Q h T
R V W G Enqueue : W, G Dequeue: t	t’’→ U→ W → G R’→ h V’→ T W’ → t G’ → t	A B K C D J X n L F E I M Y H N S P Z O U Q h T t
V W G Enqueue : Dequeue: R	R’’→ h V’→ T W’ → t G’ → t	A B K C D J X n L F E I M Y H N S P Z O U Q h T t R
W G Enqueue : Dequeue: V	V’’→ T W’ → t G’ → t	A B K C D J X n L F E I M Y H N S P Z O U Q h T t R V
G Enqueue : Dequeue: W	W’’ → t G’ → t	A B K C D J X n L F E I M Y H N S P Z O U Q h T t R V W
∞ Enqueue : Dequeue: G	G’’ → t	A B K C D J X n L F E I M Y H N S P Z O U Q h T t R V W G

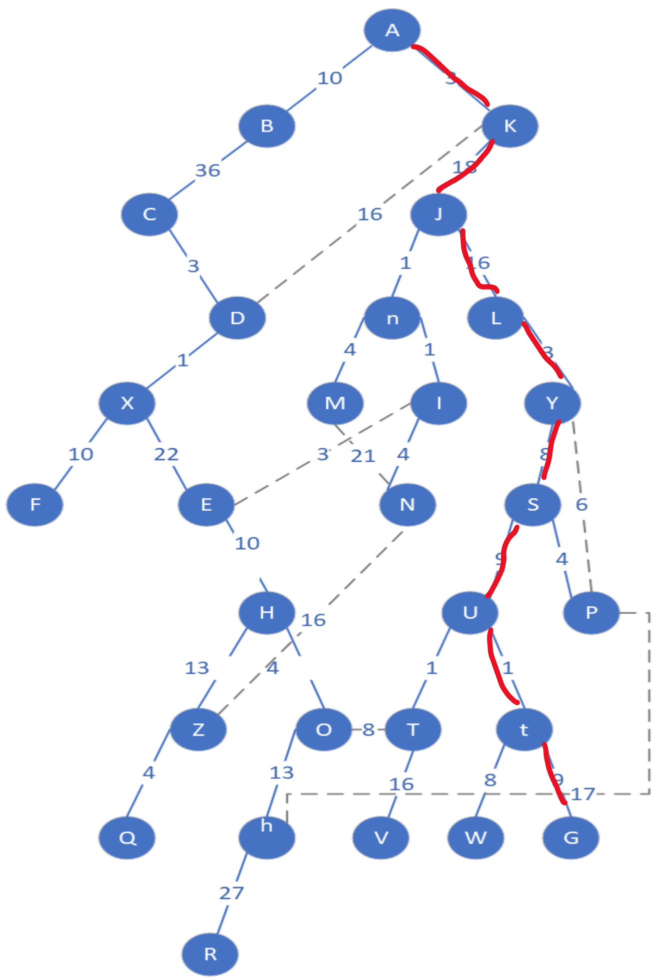
8b. Construct the corresponding breadth-first search (BFS) tree forest, with an indication of tree edges and cross edges in addition to back edges and forward edges)

ANSWER:



8c. From the obtained BFS tree forest, compute the shortest distance (smallest number of edges) from A to vertex G.

ANSWER:



From vertex A to vertex G. The traversal order should be A, K, J, L, Y, S, U, t, G (Visual it from the tree structure).

The shortest distance is the sum of AK + KJ + JL + YS + SU + Ut +tG = 3 + 18 + 16 + 3 + 8 + 9 + 1 + 9 = <u>67</u>

8d. What are the time efficiency and space efficiency of the BFS?

ANSWER:

Analyze the running time of an input graph G = (V, E):

- the total time spent in scanning adjacency lists is O(|E|)
- the total time devoted to queue operation is O(|V|)
- Therefore, the BFS run is linear-time in the size of the adjacency-list representation of G. the total running time of the BFS procedure is O(|V| + |E|).

Space efficiency is the total space that needs to store all the data structures below in computer memory:

- a data structure to store graph G (vertex and edges) (dictionary implementation)
- The color[u] to store the color of each vertex u in V (list implementation)
- The attribute π[u] is the predecessor of vertex u in V (list implementation)
- The attribute d[u] holds the distance from the source s to vertex u computed by the algorithm
- The FIFO queue data structures that containing vertex s.

Problem 9 [30 points]

From the given graph in Problem 7, given a source vertex A, use Prim's algorithm to find the minimum spanning tree for the graph. For each step, state your tree vertices VT and the remaining vertices V - VT. More importantly, you need to give a table stating the tree vertices and remaining vertices with their weights (i.e., the corresponding edges with their weights). You do not have to give the intermediate graph as an "Illustration", but you show the final minimum spanning tree (via highlight edges) within the graph given in problem 7.

9a. Compute the minimum spanning tree of the graph given in Problem 7.

Tree Vertices	Remaining Vertices	VT	V - VT
A(-, -)	B(A, 10), K(A, 3), ?(-, ∞)	{A}	{ B, K, ?}
	. . .		

where VT = { A } and

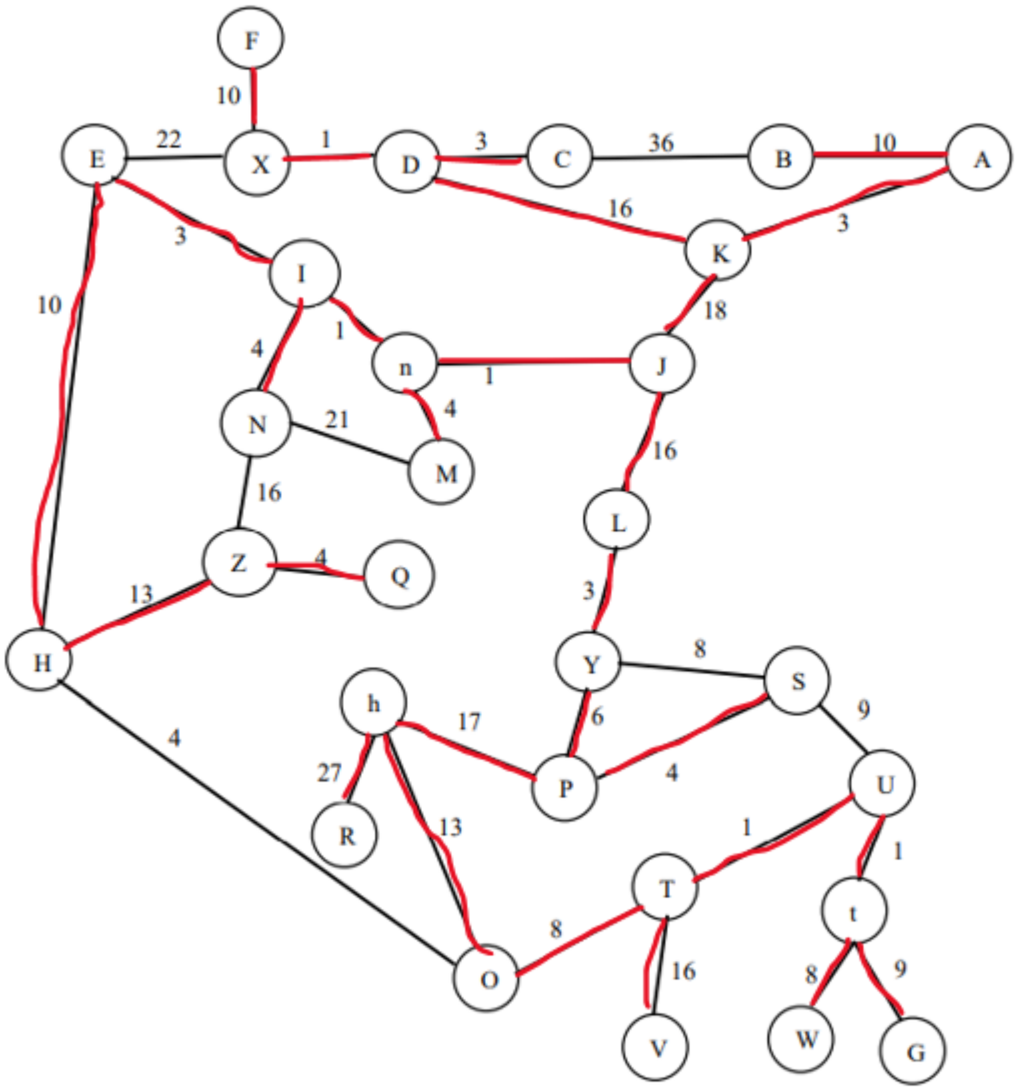
V - VT = { B, K, ?}, where “?” is to denote any vertex in the graph, which is not adjacent to A.

If you wish, use the symbol “?” to denote all the vertices in the graph which are not adjacent to every vertex in VT.

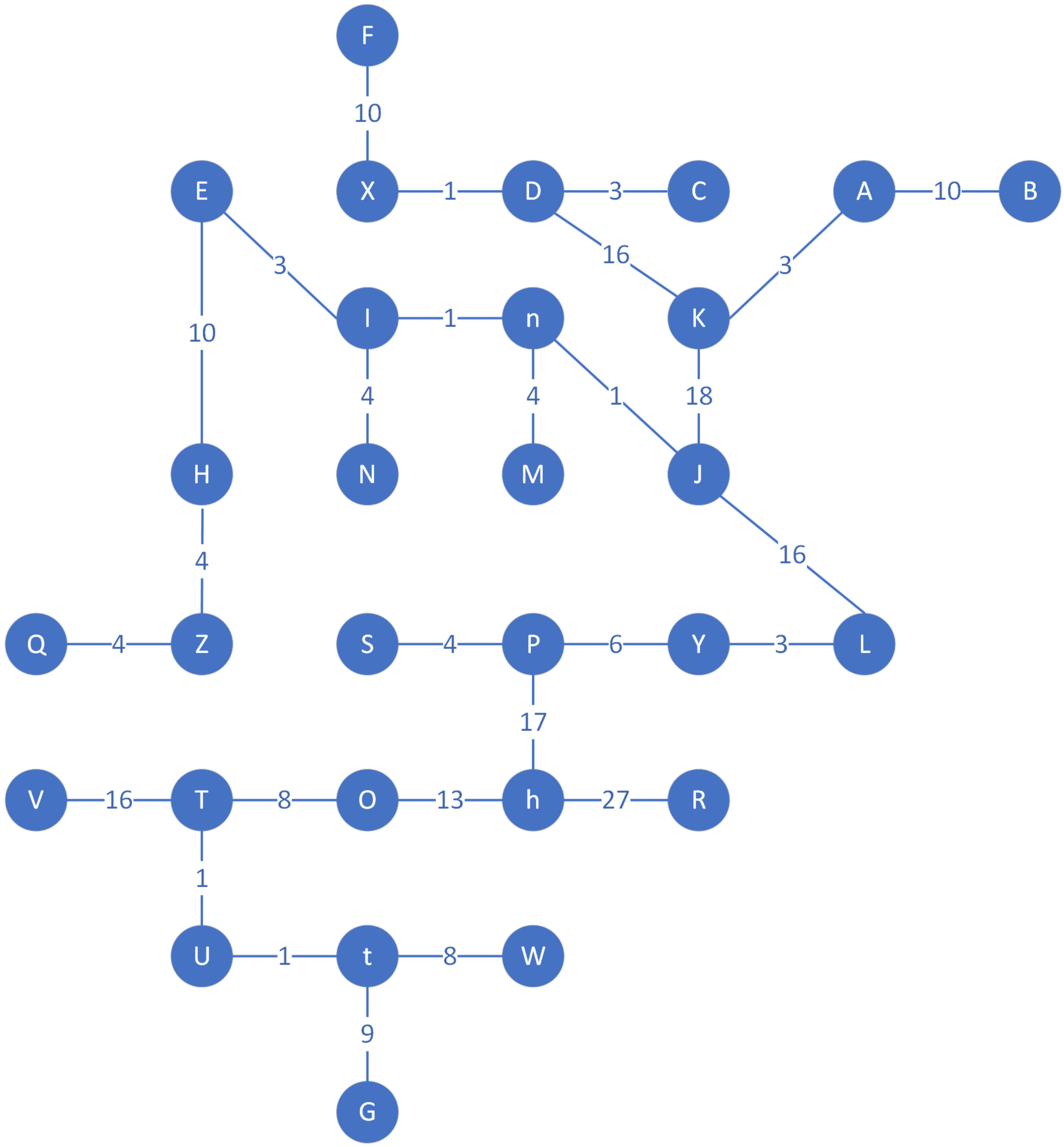
ANSWER:

Tree Vertices	Remaining Vertices (applied min-heap)	V_T	$V - V_T$
F(-, -)	X(F,10), ?(-,∞)	{F}	{ X, ?}
X(F,10)	D(X,1), E(X,22), ?(-,∞)	{F, X}	{D, E, ?}
D(X,1)	C(D,3), K(D,16), E(X,22), ?(-,∞)	{F, X, D}	{C, K, E, ?}
C(D,3)	K(D,16), E(X,22), B(C, 36), ?(-,∞)	{F, X, D, C}	{K, E, B, ?}
K(D,16)	A(K,3), J(K,18), E(X,22), B(C, 36), ?(-,∞)	{F, X, D, C, K}	{A, J, E, B, ?}
A(K,3)	B(A,10), J(K,18), E(X,22), B(C, 36), ?(-,∞)	{F, X, D, C, K, A}	{B, J, E, ?}
B(A,10)	J(K,18), E(X,22), ?(-,∞)	{F, X, D, C, K, A, B}	{J, E, ?}
J(K,18)	n(J,1), L(J,16), E(X,22), ?(-,∞)	{F, X, D, C, K, A, B, J}	{n, L, E, ?}
n(J,1)	I(n,1), M(n,4), L(J,16), E(X,22), ?(-,∞)	{F, X, D, C, K, A, B, J, n}	{I, M, L, E, ?}
I(n,1)	E(I,3), N(I,4), M(n,4), L(J,16), E(X,22), ?(-,∞)	{F, X, D, C, K, A, B, J, n, I}	{ E, N, M, L, ?}
E(I,3)	N(I,4), M(n,4), H(E,10), L(J,16), ?(-,∞)	{F, X, D, C, K, A, B, J, n, I, E }	{N, M, H, L, ?}
N(I,4)	M(n,4), H(E,10), L(J,16), Z(N,16), ?(-,∞)	{F, X, D, C, K, A, B, J, n, I, E, N }	{ M, H, L, Z, ?}
M(n,4)	H(E,10), L(J,16), Z(N,16), ?(-,∞)	{F, X, D, C, K, A, B, J, n, I, E, N, M }	{H, L, Z, ?}
H(E,10)	Z(H,13), L(J,16), Z(N,16), ?(-,∞)	{F, X, D, C, K, A, B, J, n, I, E, N, M, H}	{Z, L ?}
Z(H,13)	Q(Z,4), L(J,16), ?(-,∞)	{F, X, D, C, K, A, B, J, n, I, E, N, M, H, Z}	{Q, L ?}
Q(Z,4)	L(J,16), ?(-,∞)	{F, X, D, C, K, A, B, J, n, I, E, N, M, H, Z, Q}	{L, ?}

L(J,16)	Y(L,3), ?(-,∞)	{F, X, D, C, K, A, B, J, n, I, E, N, M, H, Z, Q, L}	{Y, ?}
Y(L,3)	P(Y,6), S(Y,8), ?(-,∞)	{F, X, D, C, K, A, B, J, n, I, E, N, M, H, Z, Q, L, Y }	{P, S ?}
P(Y,6)	S(P,4), h(P,17), S(Y,8), ?(-,∞)	{F, X, D, C, K, A, B, J, n, I, E, N, M, H, Z, Q, L, Y, P}	{S, h ?}
S(P,4)	h(P,17), ?(-,∞)	{F, X, D, C, K, A, B, J, n, I, E, N, M, H, Z, Q, L, Y, P, S}	{h, ?}
h(P,17)	O(h,13), R(h,27), ?(-,∞)	{F, X, D, C, K, A, B, J, n, I, E, N, M, H, Z, Q, L, Y, P, S, h}	{O, R, ?}
O(h,13)	T(O,8), R(h,27), ?(-,∞)	{F, X, D, C, K, A, B, J, n, I, E, N, M, H, Z, Q, L, Y, P, S, h, O}	{T, R, ?}
T(O,8)	U(T,1), V(T,16), R(h,27), ?(-,∞)	{F, X, D, C, K, A, B, J, n, I, E, N, M, H, Z, Q, L, Y, P, S, h, O, T}	{U, V, R, ?}
U(T,1)	t(U,1), V(T,16), R(h,27), ?(-,∞)	{F, X, D, C, K, A, B, J, n, I, E, N, M, H, Z, Q, L, Y, P, S, h, O, T, U}	{t, V, R, ?}
t(U,1)	W(t,8), G(t,9), V(T,16), R(h,27)	{F, X, D, C, K, A, B, J, n, I, E, N, M, H, Z, Q, L, Y, P, S, h, O, T, U, t}	{W, G, V, R}
W(t,8)	G(t,9), V(T,16), R(h,27)	{F, X, D, C, K, A, B, J, n, I, E, N, M, H, Z, Q, L, Y, P, S, h, O, T, U, t, W}	{G, V, R}
G(t,9)	V(T,16), R(h,27)	{F, X, D, C, K, A, B, J, n, I, E, N, M, H, Z, Q, L, Y, P, S, h, O, T, U, t, W, G}	{V, R}
V(T,16)	R(h,27)	{F, X, D, C, K, A, B, J, n, I, E, N, M, H, Z, Q, L, Y, P, S, h, O, T, U, t, W, G, V}	{R}
R(h,27)		{F, X, D, C, K, A, B, J, n, I, E, N, M, H, Z, Q, L, Y, P, S, h, O, T, U, t, W, G, V, R}	{}

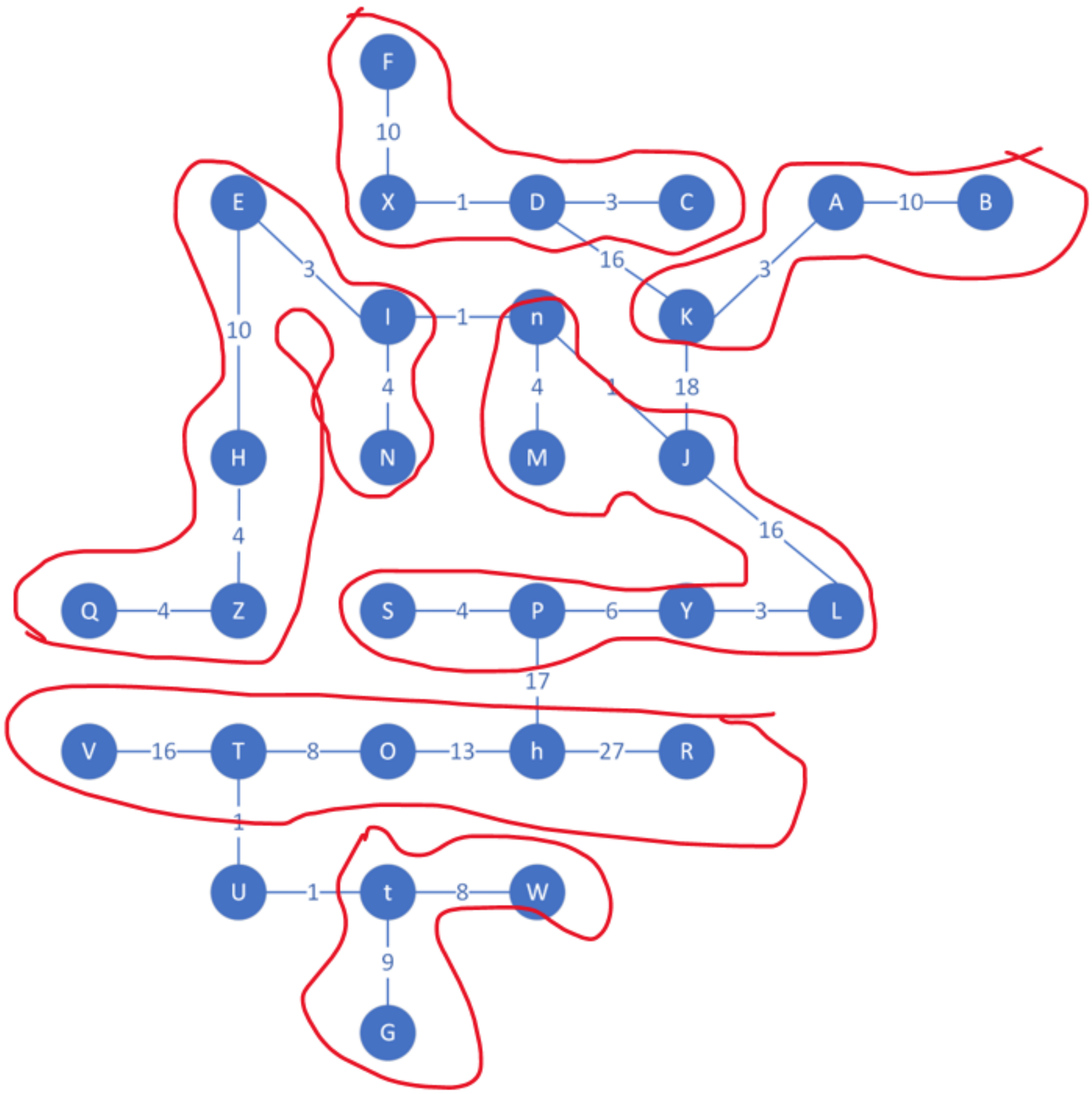


Here is a nice diagram of the above using visual studio [Drawing.vsdx]



9b. What is your obtained minimum spanning tree with their total weights of branches for the graph given in problem 7? [Highlight the obtained (from 9a) minimum spanning tree in the given graph of Problem 7. For example, the branch of A, K, D, X, F has a weight Secondly, compute the total weight of each branch of the minimum spanning tree. Thirdly, compute the grand total weight of the obtained minimum spanning tree.]

ANSWER:



- The grand total minimum weight of the spanning tree is: 230

Branch	Weight
F, X, D, C	14
K, A, B	13
Q, Z, H, E, I, N	25
M, n, J, L, Y, P, S	34

V, T, O, h, R	64
G, t, W	17

9c. From your obtained minimum spanning tree, what is the minimum distance from vertex A to vertex G?

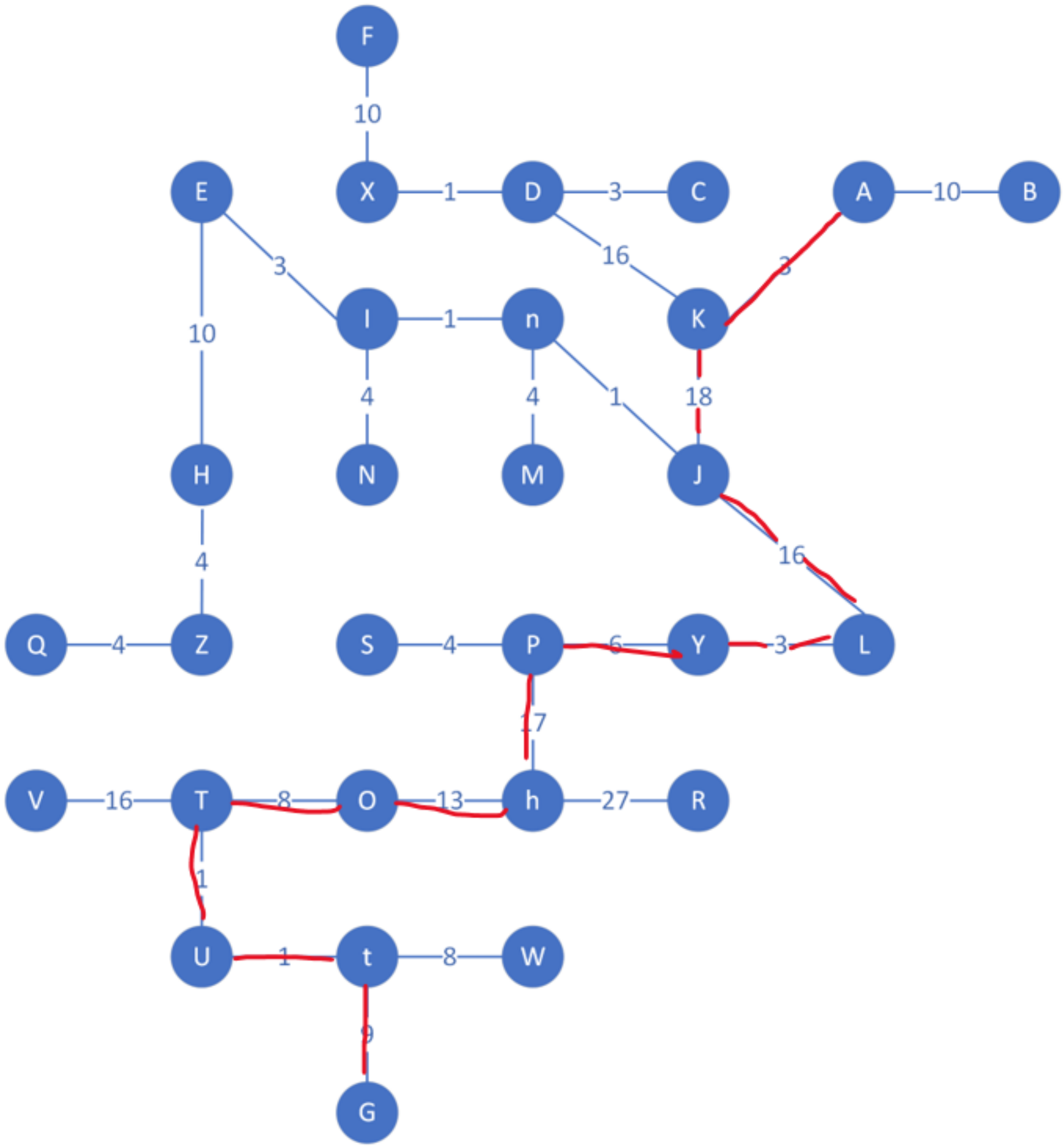
ANSWER:

The shortest path from vertex A to vertex G is following through the below vertex in the following order:

A, K, J, L, Y, P, h, O, T, U, t, G

The minimum distance is the sum of each edge between these vertexes. It is 95

Please refer to the below diagram for verification



Note: Good handwriting is required if you provide your answer in your handwriting. Proper numbering of your answer to each problem is strictly required. The problem's solution must be orderly given. (10 points off if not)

CS 58000_01 Algorithm Design, Analysis & Implementation (3 cr.)

Assignment As_01

Student Name: Truc Huynh

I: [80 points]: each of I.1 through I.8 is 10 points.

(I.1) Give the input and output specifications for :

(a) the algorithm a?
(b) the algorithm b?

Answer:

	algorithm a	algorithm b
Input	Two n-bit integer x and y, where $y \geq 1$	Two n-bit integer x and y, where $y \geq 1$
Output	The quotient and reminder of x divided by y	The quotient and reminder of x divided by y

(I.2) What is the input size for:

(a) the algorithm a?
(b) the algorithm b?

Answer:

	algorithm a & algorithm b
Input Size	Algorithm a & algorithm b will have the same input size as they take x and y for division The input x or y will be an integer n. The input size for each of them will be $\lfloor \log2\ n \rfloor$ + 1 bits Define the input size as the number of symbols (in this case bits) used for encoding a positive integer n.

(I.3) What is the basic operation for:

(a) the algorithm a?
(b) the algorithm b?

Answer:

	algorithm a	algorithm b
Basic Operation	Right shift (bit additions)	Left shift (bit multiplication) Right shift (bit additions)

(I.4) In algorithm b, what is the functionality of the following segment of statements? (i.e., give the reasons for writing each of the statements. Why double the q and r? Why increase r by one when x is odd? Why reduce r by y and increase q by one when r ≥ y?)

q := 2 * q; r := 2 * r;

if (x is odd) then r := r + 1;

if (r ≥ y) then

{ r := r – y; q := q + 1};

Answer:

Why double the q and r?	In the recursive call, we divide x by 2. There we must double q and r to make it even. It can be done by shift left one bit
Why increase r by one when x is odd?	Because we want to get the top division (not the floor division) when using integer division. We want to change it to an even number that can divide by 2. Ex: 7/2 = 3 (floor division) but that is not what we want to get. Therefore, we add (7+1)/2 = 4
Why reduce r by y and increase q by one when r ≥ y?)	- Reduce r by y to get the remaining value of x - Adding 1 to q because we just subtract y from (reduce) the value of r. Thus, we prove that previous r (before the subtraction) can divide one more y -> Increase q by 1

(I.5) Analyze and derive the algorithm's time and space efficiency for Algorithm a.

(Hint: express time efficiency in terms of summation $\sum_{\square}^{\square} \square$)

Given algorithm a,

if x = 0, then return (q, r) := (0, 0);

q := 0; r := x;

while (r ≥ y) do   // takes n iterations for the worse case.

{ q := q + 1;

r := r – y}; // O(n) for each r – y, where y is n bits long.

return (q, r);

Answer:

	algorithm a

Time Efficiency	Takes 2^n iterations for the worse case, which says x/1 where x is n bits long with a max value $2^n – 1$. Since this takes linear time O(n) for each iteration. The worse case would be $O(n2^n)$
Space Efficiency	Take 4 spaces: x, y, q, r

(I.6) Analyze and derive the algorithm's time and space efficiency for algorithm b.

(Hint: for time efficiency, use recurrence relation and solve for the recurrence relation system)

Given algorithm b,

```
if (x = 0) then return (q, r):=(0, 0);
(q, r) := divide(⌊x/2⌋ , y )        //requires n-bits right shift
q := 2 * q,  r := 2 * r;            // shift left one bit.
if (x is odd) then r := r + 1;      // needs c*n-bits
if (r ≥ y) then                     // additions
        { r := r – y; q := q + 1};
return (q, r);
```

Answer:

	algorithm a
Time Efficiency	- Takes n call, so the worse case would be O(n2)
Space Efficiency	- Take 4 spaces: x, y, q, r

(I.7) Can these two algorithms a and b be improved? Justify your answer.

(a) for the algorithm a?
(b) for the algorithm b?

Answer:

	algorithm a	algorithm b
Improvement	- No improvement if the use of iteration is applied. - Can only improve by using recursive (algorithm b) because time complexity is huge compared to algorithm b $(O(n2^n) > O(n^2))$ - x must be ≥ 0 for the input (precondition).	- Pretty optimal already (in terms of time and space complexity). - x must be ≥ 0 for the input (precondition).

(I.8) Compare these two algorithms a and b:

Which is a better algorithm in terms of time and space efficiency? Justify your answer.

Answer:

	algorithm a	algorithm b
Time efficiency	- $O(n2^n)$ is slower algorithm b	- $O(n^2)$ is faster than algorithm a
Space efficiency	- Take 4 spaces: x, y, q, r. The calculation between them is stored in the computer register (the number of registers depends on the computer). Thus, the same efficiency as algorithm b.	- Take 4 spaces: x, y, q, r. Thus, the same efficiency as algorithm a

Reference:

Purdue slides Ch000_02_IntroFoundation_ProgCorrectionLec.pdf from CS 58000_01 Algorithm Design, Analysis & Implementation

CS 58000_01 Algorithm Design, Analysis & Implementation (3 cr.)

Assignment As_01

Student Name: Truc Huynh

80 – 28 = 52

Why is your filename an ….CS486_As01___36?????

If this was your solution for CS 486, then I would say that you did not improve the analysis of algorithms.

I: [80 points]: each of I.1 through I.8 is 10 points.

(I.1) Give the input and output specifications for :

(a) the algorithm a?
(b) the algorithm b?

Answer:

	algorithm a	algorithm b
Input	Two n-bit integer x and y, where $y \geq 1$	Two n-bit integer x and y, where $y \geq 1$
Output	The quotient and reminder of x divided by y	The quotient and reminder of x divided by y

-4

Answer: For both a and b, their input and output specifications are as follows:
Input specification: {x, y | x and y are two integers, where **$x \geq 0$, $y \geq 1$**}.
Output specification: {q, r | x = q*y + r, where **$0 \leq r < y$**}. q and r are the quotient and remainder for y dividing x.

(I.2) What is the input size for:

(a) the algorithm a?
(b) the algorithm b?

Answer:

	algorithm a & algorithm b
Input Size	Algorithm a & algorithm b will have the same input size as they take x and y for division The input x or y will be an integer n. The input size for each of them will be $\lfloor \log 2\ n \rfloor + 1$ bits Define the input size as the number of symbols (in this case bits) used for encoding a positive integer n.

Answer: For both (a) and (b), their input size is:

The number of bits for representing the value of x, say m. $m = \lfloor \log_2 x \rfloor + 1$.
The number of bits for representing the value of y, say n. $n = \lfloor \log_2 y \rfloor + 1$.

Not n but x and y. -3

(I.3) What is the basic operation for:

(a) the algorithm a?
(b) the algorithm b?

Answer:

	algorithm a	algorithm b
Basic Operation	Right shift (bit additions)	Left shift (bit multiplication) Right shift (bit additions)

For (a): The basic operation is the **compare operation (r ≥ y),** or "either + or :=" in "(r := r + -y); or q := q + 1;".

For (b): The basic operation could be: compare operation (x == 0) or **integer division ⌊ x/2 ⌋** (shift x right by one bit) for each recursive call. Both can be used to define the number of recursive calls.

-4

(I.4) In algorithm b, what is the functionality of the following segment of statements? (i.e., give the reasons for writing each of the statements. Why double the q and r? Why increase r by one when x is odd? Why reduce r by y and increase q by one when r ≥ y?)

```
q := 2 * q;  r := 2 * r;
if (x is odd) then r := r + 1;
if (r ≥ y) then
    { r := r – y; q := q + 1};
```

Answer:

Why double the q and r?	In the recursive call, we divide x by 2. There we must double q and r to make it even. It can be done by shift left one bit
Why increase r by one when x is odd?	Because we want to get the top division (not the floor division) when using integer division. We want to change it to an even number that can divide by 2. Ex: 7/2 = 3 (floor division) but that is not what we want to get. Therefore, we add (7+1)/2 = 4
Why reduce r by y and increase q by one when r ≥ y?)	- Reduce r by y to get the remaining value of x - Adding 1 to q because we just subtract y from (reduce) the value of r. Thus, we prove that previous r (before the subtraction) can divide one more y -> Increase q by 1

-6

Suppose that preparing for the first recursive call,

$\lfloor x/2 \rfloor = \mathbf{q_0 * y + r_0}$ ……… (1)

Then, where x is even, $\lfloor x/2 \rfloor$ = $\mathbf{x/2 = q_0 * y + r_0}$, since x is divisible by 2.

$\mathbf{x = 2q_0 * y + 2r_0}$; …….(2)

where x is old, $\lfloor x/2 \rfloor$ = $\mathbf{(x-1)/2 = q_0 * y + r_0}$

$\mathbf{x = 2q_0 * y + 2r_0 + 1}$ …….(2)

From (2), $q_1 = 2q_0$ and $r_1 = 2r_0$, then $q_{i+1} = 2q_i$ and $r_{i+1} = 2r_i$;

these lead to "q := 2 * q, r := 2 * r;",

Preparing for the i^{th} recursive call, (2) implies (2'),

$x = 2q_{i-1} * y + 2r_{i-1}$, where x is even,

or $x = 2q_{i-1} * y + 2r_{i-1} + 1$, where x is old. …….(2')

That is, after the first recursive call,

$r_1 = 2r_0$, where x is even,

or $r_1 = 2r_0 + 1$, where x is old. …………..(3)

(3) implies (3'):

That is, after ith recursive call,

$r_i = 2r_{i-1}$, where x is even,

or $r_i = 2r_{i-1} + 1$, where x is old. …………..(3')

This leads to "if (x is odd) then r := r + 1; ", where x is odd.

However, cases would arise: either if ($r_1 < y$)

or, if ($r_1 \geq y$).

According to if ($r \geq y$) then

{ r := r – y; q := q + 1};

For if ($r_1 < y$) then no action "{ r := r – y; q := q + 1};" is needed to be taken.

For if ($r_1 \geq y$), this implies that 2y is ≤ r_0. Since ($r_1 \geq y$) then

{ $r_2 := r_1 - y$; $q_2 := q_1 + 1$};

implying that if ($r_i \geq y$) then

$\{ r_{i+1} := r_i – y; q_{i+1} := q_i + 1\};$ ….. (4)

For if ($r_1 \geq y$) then execute "{ r := r – y; q := q + 1};"; reduce r_1 by one times of y to generate $r_2 = r_1 – y;\ 0 \leq r_2 < r_1$. and $q_2 := q_1 + 1$. That means $x = q_2 * y + r_2$. That is, $r_2 = x – q_2*y$.

This leads to "there exists an integer $i \geq 1$, such that $0 \leq r_i – i*y < y$.

Equivalently, there exists an integer $i \geq 1$, such that $0 \leq i*y \leq r_i < (i + 1)y$.

Assume that these are true for i, "there exists an integer $i \geq 1$, such that $0 \leq r_i – i*y < y$."

if ($r_i < y$) is true, no further execution of "{ r := r – y; q := q + 1};" is needed. Otherwise,

if ($r_i \geq y$) is true, then $r_{i+1} = r_i - y$, and $q_{i+1} := q_i + 1$, such that $x = q_{i+1} * y + r_{i+1}$.

if $x = q_{i+1} * y + r_{i+1}$, then $x = (q_i + 1) * y + (r_i – y)$ using (4).

$$x = (q_i *y + 1 *y) + (r_i – y) = q_i *y + r_i$$

Therefore, if ($r_i \geq y$) then $r_{i+1} = r_i - y$; and to keep the equations (2) hold, $q_{i+1} := q_i + 1$ for each of the subtract y from r. That is, this leads to

"if ($r \geq y$) then { r := r – y; q := q + 1};"

(I.5) Analyze and derive the algorithm's time and space efficiency for Algorithm a.

(Hint: express time efficiency in terms of summation $\sum_{\square}^{\square} \square$)

Given algorithm a,

```
if  x = 0, then return (q, r) := (0, 0);
        q := 0;  r := x;
while (r ≥ y) do      // takes n iterations for the worse case.
        { q := q + 1;
        r := r – y};  // O(n) for each r – y, where y is n bits long.
return (q, r);
```

Answer:

	algorithm a
Time Efficiency	Takes 2^n iterations for the worse case, which says x/1 where x is n bits long with a max value $2^n – 1$. Since this takes linear time O(n) for each iteration. The worse case would be $O(n2^n)$
Space Efficiency	Take 4 spaces: x, y, q, r

Consider the worst-case, name y = 1. Assume that x is n + 1 bits long. The position of bit at n is a sign bit. The maximum value of x would be $2^n - 1$. Then the value of x will take $(2^n - 1)$ time reducing the value of x by 1 to get down to 0. That is, $(2^n - 1)$ number of times executing ($r \geq y$). For a large n, $(2^n - 1) \approx 2^n$. That means it requires 2^n time for executing the '+' addition operation for the statement r := r + (-y); and likewise 2^n time for executing the '+' addition operation for the statement q := q + 1;

Since each addition operation requires O(n), the number of times for an addition operation could be n 2^n for executing r = r – y in the **worse case; which is exponential, $O(n\ 2^n) = O(2^n)$**

Likewise for the best case, either x = 0, or $x \neq 0$ and x < y, or if $x \neq 0$ and $x \geq y$, where x = y, then **Ω(1), for the best case.**

For **the average case, it is $O(2^n)$.**

For space efficiency,
it requires space for each of the variables x, y, q, and r. It requires a constant c space. Or it requires cn bits, where n is the maximum bits for the max{x, y, q, r} bits. **That is, space efficiency is Θ(1).**

-4

(I.6) Analyze and derive the algorithm's time and space efficiency for algorithm b.

(Hint: for time efficiency, use recurrence relation and solve for the recurrence relation system)

Given algorithm b,

```
if (x = 0) then return (q, r):=(0, 0);
(q, r) := divide(⌊x/2⌋, y )       //requires n-bits right shift
q := 2 * q,  r := 2 * r;          // shift left one bit.
if (x is odd) then r := r + 1;    // needs c*n-bits
if (r ≥ y) then                   // additions
        { r := r – y; q := q + 1};
return (q, r);
```

Answer:

	algorithm a
Time Efficiency	- Takes n call, so the worse case would be O(n2)
Space Efficiency	- Take 4 spaces: x, y, q, r

For time efficiency

Assume $x \geq y$ where x is n bits long and $n = \lfloor \log_2 x \rfloor + 1$.
If x = 0, then one compare operator (=) and two := operations

If $x \neq 0$ then it has $n = \lfloor \log_2 x \rfloor + 1$ number of recursive calls. For each call, there is a shift right one bit on x. **That means, there will be n numbers of recursive calls**. Upon return from each recursive call, it requires two := operations for (q, r).

For each recursive, upon each of the returns, it needs two numbers of left shift on q and r. two := operations from q := **2 * q**, r := **2 * r;** Since it has n times of recursive call, then it requires 2 n shift left of q and r, and 2 n of := operation. **That means these require each O(n) for each recursive call.**

if (x is odd) then r := r + 1; To determine whether (x is odd) by examine the rightmost digit of x whether is 1 (for x is odd) or 0 (for x is even). And, if x is odd, then it requires one + and one := operations. Since it has n times of recursive call, each of them at most would be executed n times. **That is, this requires O(n) for each recursive call.**

Then one $r \geq y$ compare operation, and if $r \geq y$, then execute two + and two := operations **for { r := r – y; q := q +1} statements**. We can assume that **they require O(n) for each recursive call.**

Since there is an n number of the recursive call, and upon each return, it requires O(n) for executing the rest of the statements:

$$n(O(n)) = O(n^2), \text{ where } n = \lfloor \log_2 x \rfloor + 1.$$

For space efficiency,

It requires space for x, y, q, r, and space cn for the return address after each recursive call. It has **Θ(n).** space for a return address. Otherwise, it is a constant **Θ(1).**

-3

(I.7) Can these two algorithms a and b be improved? Justify your answer.

(a) for the algorithm a?
(b) for the algorithm b?

Answer:

	algorithm a	algorithm b
Improvement	- No improvement if the use of iteration is applied. - Can only improve by using recursive (algorithm b) because time complexity is	- Pretty optimal already (in terms of time and space complexity). - x must be ≥ 0 for the input (precondition).

	huge compared to algorithm b ($O(n2^n) > O(n^2)$) - x must be ≥ 0 for the input (precondition).	

Answer:

(a) Since **algorithm b is an improvement when compared to algorithm a in terms of time**. The algorithm b **reduces an exponent time** from algorithm a to **quadrative time,** despite the algorithm b having O(n) space complexity compared to the O(1) for algorithm a.

(b) For some cases, **algorithm b could be reduced to O(1) space complexity**.

-2

(I.8) Compare these two algorithms a and b:

Which is a better algorithm in terms of time and space efficiency? Justify your answer.

Answer:

	algorithm a	algorithm b
Time efficiency	- $O(n2^n)$ is slower algorithm b	- $O(n^2)$ is faster than algorithm a
Space efficiency	- Take 4 spaces: x, y, q, r. The calculation between them is stored in the computer register (the number of registers depends on the computer). Thus, the same efficiency as algorithm b.	- Take 4 spaces: x, y, q, r. Thus, the same efficiency as algorithm a

Answer

Let n = = $\lfloor \log_2 x \rfloor$ + 1 bits. For the **worse case**, algorithm a requires $O(n2^n)$ that grows exponentially, but algorithm b requires only $O(n^2)$. In the **best case**, both algorithms could be achieved by constant time O(1). For the **average case,** algorithm a is $\Theta(2^n)$ and algorithm b is $\Theta(n^2)$

-2

Reference:

Purdue slides Ch000_02_IntroFoundation_ProgCorrectionLec.pdf from CS 58000_01 Algorithm Design, Analysis & Implementation

CS 58000-01 Quiz01

August 30, 2022 Print your name ___________________________________
First Name Last Name

1. Problem: Write a most time-efficient sequential search algorithm to search for a given value *x* in a given array S[0 .. n-1].

   Input Specifications: An array S[0 .. *n*-1] of n elements and a search key *x*.

   Output Specification: The index (location) of the first element of *S* that matches *x* or -1 if there are no matching elements.

2. Give an instance of the given problem in 1.

Soln:

1.

S[n] := x;

i := 0;

while (S[i] $\neq$ x)

do {i := i + 1; }

if (i < n) return i;

else return -1;

2.

An instance of the problem can be as follows:

S[0] = 14 x = 16

S[1] = 11

S[2] = 15

S[3] = 20

CS 58000-01 Quiz 02 Print your Name: ________________________________

September 6, 2022

1. Analyze the time efficiency for the function multiply(x, y).

   function multiply(x, y)

   Input: Two integers x and y, where $y \geq 0$
   Output: Their product
   if $y = 0$ then return 0;
   z := multiply (x, $\lfloor y/2 \rfloor$);
   if y is even then return 2z
   else return x + 2z;

   a. What is the input size of any integer y?
   - __ $n = \lfloor \log_2 y \rfloor + 1$ bits; e.g., let $2^7 \leq y < 2^8$. Then n is 8 bits long. ___

   b. What is the total number of recursive calls "multiply (x, $\lfloor y/2 \rfloor$" executed?

      ___At most $n = \lfloor \log_2 y \rfloor + 1$ number of recursive calls _____________

   c. Upon each return from the recursive call "multiply (x, $\lfloor y/2 \rfloor$)", assign the product to z, and then execute the if-then-else statement. What is the time efficiency for executing this if-then-else statement?

      _________O(n) justifying by the following problems d (O(n)), e(O(1)), and f (O(n)).

      ____

   d. What is time efficiency for determining whether y is even?

      ______constant time O(1) for every testing the rightmost bit whether is a zero. The other way is to determine whether ($y - (\lfloor y/2 \rfloor)*2$) is zero, where $\lfloor y/2 \rfloor)*2$ can be done by shift-right y by one bit and then shift-left by one bit. Then add the negative result to y. If it is zero, then y is even. This require O(n) + O(1) + O(1) = O(n)_______

   e. What is time efficiency for computing 2z?

      ______ constant time O(1) for each left shift-left z by one bit and then append a zero at the rightmost bit, for any given z.______

   f. What is time efficiency for computing x + 2z?

      _______In addition to O(1) for computing 2z, each addition requires O(n), assuming x and y are n-bits long. The time efficiency is O(n)__

   g. What is the time efficiency of this algorithm, function multiply(x, y)?

      _______$n * O(n) = n* (c_0 + c\, n) = O(n^2)$.________________________

h. Express the function multiple(x, y) in terms of an equation system.

$$x * y = \begin{cases} 2(x * \lfloor y/2 \rfloor), & \text{if y is even} \\ x + 2(x * \lfloor y/2 \rfloor), & \text{if y is odd} \end{cases}$$

i. Is the function multiple(x, y) correct?

It is transparently correct; It also handles the base case (y = 0).________________

j. Does the function multiple(x, y) halt?

_ The function multiple(x, y) will eventually halt. The reason is: For each recursive call multiply (x, $\lfloor y/2 \rfloor$), the continuation of "shift-right y by one bit" for a given $n = \lfloor \log_2 y \rfloor + 1$ bits will eventually yield zero for the value of y. The recursive call will be ended if y = 0 then return 0;________________________

CS 58000-01/2I Quiz 03 Print your Name: Truc Huynh

September 15, 2022

10 points

Analyze the time efficiency for the function modexp(x, y, N)

function modexp(x, y, N)

//Compute x^y mod N

Input: Two n-bit integers x and N, and an integer exponent y.

Output: x^y mod N.

if (y = 0) then return 1;

z = modexp(x, $\lfloor y/2 \rfloor$, N); // z = $x^{\lfloor y/2 \rfloor}$ mod N

if (y is even) then return z^2 mod N;

else return x * z^2 mod N;

1. What is the basic operation?

   Answer:

   The basic operation would be to integer division, exponent a number, and multiply two mod N numbers

2. What is the input size

   Answer:

   Since there are 2 n-bits integer x and N, and an integer exponent y. Therefore, each of these numbers x, y, and N has the size n = $\lfloor \log_2 y \rfloor + 1$ bit.

3. How many times of recursive calls will be made?

   Answer:

   The algorithm will halt at $\log_2 n$ (since $\lfloor y/2 \rfloor$ will shilt one bit after each call) recursive call.

4. What is the best way to compute z^2 mod N:

   Answer:

We have $z = x^{\lfloor y/2 \rfloor}$ mod N. Therefore, z^2 mod N = $(x^{\lfloor y/2 \rfloor}$ % N$)^2$ % N. We make the z which is $x^{\lfloor y/2 \rfloor}$ % N to the power of 2, which is splitting the y (or y/2) by the power of 2. So, we split the ((x^y) % N) Until $x^y = x^3$. Then, $x^3 = x * (x^1)^2$ By using these formulas below we can prove it:

$x^{2n} = (x^n)^2$ for all real numbers x and a with $x \geq 0$.

E.g: x^4 % n = $(x^2)^2$ % n = $(x^2$ % n$)^2$ % n

$x^{a+b} = x^a x^b$ for all real numbers x, a and b with $x \geq 0$.

E.g.: x^7 % n = (x^{4+2+1}) % n = $(x^4x^2x^1)$ % n = {(x^4 % n) (x^2 % n)(x^1 % n)} % n

To clarify my point. Let's look at the example:

1. modexp (x, $\lfloor 12/2 \rfloor$, N); $y = 6 \neq 0$

   Therefore z^2 % N = $(x^{\lfloor 12/2 \rfloor}$ % N$)^2$ %N

   = $(x^6$ % N$)^2$ %N

   = $((x^3)^2$ % N$)^2$ %N

   = $((x^3$ % N$)^2$ % N$)^2$ %N

   = $(((x^{2+1})$ % N$)^2$ % N$)^2$ %N

   = $(((x * x^2)$ % N$)^2$ % N$)^2$ %N

   = $(x * ((x * 1^2)$ %N$)^2$ %N $)^2$ %N

5. What is the best way to compute $x * z^2$ mod N:

Answer:

Very similar to the above answer. We have $z = x^{\lfloor y/2 \rfloor}$ % N. Therefore, $x * z^2$ % N = $x * (x^{\lfloor y/2 \rfloor}$ % N$)^2$ % N (y is an integer exponent y). We make the z which is $x^{\lfloor y/2 \rfloor}$ % N to the power of 2, which is splitting the y (or y/2) by the power of 2. So, we split the ((x^y) % N) Until $x^y = x^3$. Then, $x^3 = x * (x^1)^2$ By using these formulas in answer 3 we can prove it. Let's look at the example:

1. modexp (x, $\lfloor 14/2 \rfloor$, N); y = 7 ≠ 0

Therefore $x * z^2 \% N = x * (x^{\lfloor 14/2 \rfloor} \% N)^2 \% N$

$= x * (x^7 \% N)^2 \% N$

$= x * ((x^3)^2 \% N)^2 \% N$

$= x * ((x^3 \% N)^2 \% N)^2 \% N$

$= x * (((x^{2+1}) \% N)^2 \% N)^2 \% N$

$= x * (((x * x^2) \% N)^2 \% N)^2 \% N$

$= x * (x * ((x * 1^2) \% N)^2 \% N)^2 \% N$