Quiz 3 CSCI 333 Fall 2001

5 December, 2001

Problem 1 (15 points)

Write a small bit of C++ (or C) code to read a single byte of data stored at byte position 3000 of the file `QUIZ3.DAT` and write that byte back to position 1000 of the same file.

```
char byteBuff ;
// open the file for I/O
fstream updStream(FILNAME,
          ios::in | ios::out | ios::binary) ;
// read byte at position 3000
updStream.seekg(3000, ios::beg) ;
updStream.read(&byteBuff, 1) ;
// write byte to position 1000
updStream.seekp(1000, ios::beg) ;
updStream.write(&byteBuff, 1) ;
```

Problem 2 (20 points)

Assume you have a 10-slot hash table (the slots are numbered 0 through 9). Show the final hash table that would result if you used the hash function

H(k) = k mod 10

and linear probing on this list of numbers:

20, 27, 33, 37, 39

After inserting the key with value 39, list for each empty slot the probability that it will be the next one filled.

This problem is based on the textbook's Problem 9.14.

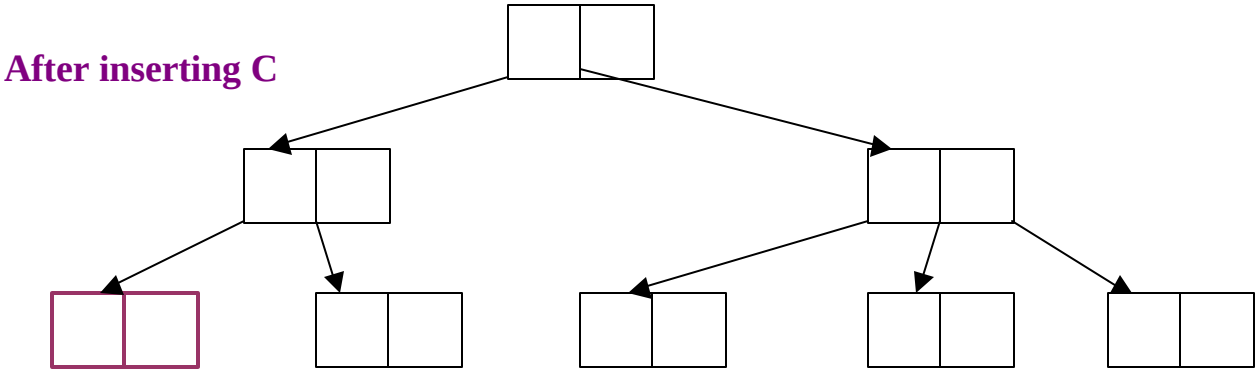
0	**20**
1	**50%**
2	**10%**
3	**33**
4	**20%**
5	**10%**
6	**10%**
7	**27**
8	**37**
9	**39**

Problem 3 (20 points)

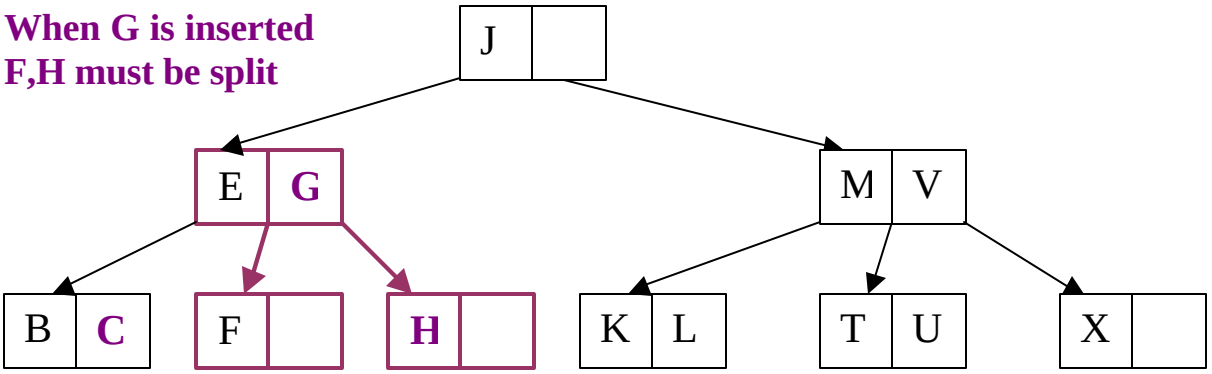
Add the following elements to the 2-3 tree shown below

C, G, A

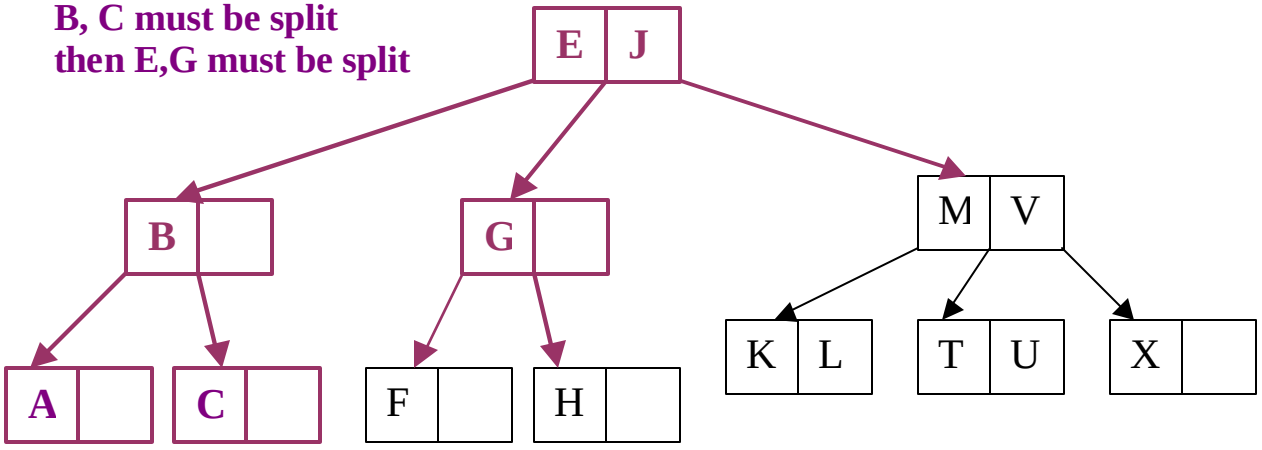
After inserting C



When G is inserted
F,H must be split



When A is inserted
B, C must be split
then E,G must be split



Problem 4 (10 points)
Consider the following two possible hash functions for mapping arrays of *exactly* four characters into a number between 0 and 2000.

```
int H(char *S) {
  int r = 1 ;
  int i ;
  for (i = 0; i < 4; ++i)
    r = r * (int)S[i] ;
  return r % 2001 ;
}
```

```
int H(char *S) {
  int r = 0 ;
  int i ;
  for (i = 0; i < 4; ++i)
    r = 3*r + (int) S[i] ;
  return r % 2001 ;
}
```

Which of these do you think would be the "better" hash function? Explain your reasoning.

Before the mod (%) the right hash can generate numbers between 0 and 255^4, or 4228250625; while the left can only generate numbers between 0 and $(3^3 + 3^2 + 3 + 1)*255$, or 23970. However, since the mod is made with 2001, which is significantly less then 23970, this advantage is negated.

In terms of raw computation speed, the two are almost indistinguishable. The right hash avoids one addition per loop, but the left has performs it multiplication by a constant (3) which could be done by two adds and a shift.

The main problem with the right hash is that it has some significant clustering problems. First of all, two words that differ only in the exchange of two letters, such as 'ACNE' and 'CANE', will hash to the same value. A worse problem occurs between 2001 is not prime, but equal to 3*23*29. The probability of a random character being divisible by 3 is 1/3. The probability that at least one of four random characters is divisible by 3 is $(1-(2/3)^4)$, approximately .8025. This means that about 80% of the possible 4-character arrays will be clustered into one-third of the hash entries.

If you want to look at this problem computationally, view the output of a C++ program that tests a wide range of possible hash inputs that has been stored into an Excel spreadsheet.

Problem 5 (15 points)
In this question, you'll give the results of executing C++ statements very similar to those found in the `ELFhash` algorithm (p. 306).

Assume that the integer variable `x` has the value `24`, or `0x18` in hex, and the integer variable `y` has the value `41`, or `0x29` in hex. In the table below, a C++ statement appears in the leftmost column. Write in the rightmost column the value assigned to x by that C++ statement. You can give your answer as either a decimal or hexadecimal number.

I've filled in the first two rows as an example of how the table should be completed.

`x = x + y ;`	65
`x = x | y ;`	0x39
`x = (x << 4) + y ;`	**(0x18 << 4) + 0x29** **0x180 + 0x29** **0x1A9** **425**
`x = y & 0xF0000000L ;`	**(0x29 & 0xF0000000)** **0**
`x ^= y >> 24 ;`	**0x18 ^ (0x29 >> 24)** **0x18 ^ 0** **0x18** **24**
`x &= ~y ;`	**0x18 & ~0x29** **0x18 & 0xFFFFFFD7** **0x10** **16**

Problem 6 (20 points)
Suppose a disk has the following characteristics:

1) Average track seek time of 20 ms
2) Track-to-track seek time of 2 ms
3) Disk rotation rate of 9000 rpm
4) Tracks containing 200 512-byte sectors
5) 10 platters
6) 3000 tracks

On the average, how long will it take to read 50 consecutive sectors stored on an arbitrary track of the disk?

Be sure to show your work.

The 50 consecutive sectors are stored on one track. Since there are 200 sectors on a track, one-fourth of a track rotation will be required to read the 50 sectors. This is in addition to the average time required to seek to the correct track and the average rotation time, one-half of track rotation, required to position the head on the first sector. So the required time is:

Seek time + Rotational delay + Transfer time
Seek time + ½Rotation time + ¼ Rotation time
Seek time + ¾Rotation time
20 ms + ¾(1/9000 minutes)
20 ms + ¾(60/9000 s)
20 ms + ¾(1/150 s)
20 ms + 1/200 s
20 ms + 5 ms
25 ms