How Big, How Fast

Thorne Melcher   
2014/02/06

**Roughly how many binary digits (bit) are required for the unsigned representation of**

1,000 – 10  
1,000,000 – 20  
1,000,000,000 – 30  
1,000,000,000,000 – 40  
8,000,000,000,000 – 43

2^3=8 and 2^4=12, therefore one extra zero takes somewhere between 3-4 bits

**My town has approximately 20,000 residences. How much space is required to store the names, addresses, and a phone number for all of these (if we store them as characters)?**

20,000 addresses \* ~.5KB/address = 10,000 KB or about 10 MB

**I’m storing 1,000,000 integers in a binary tree. Roughly how many nodes and levels can I expect the tree to have? Roughly how much space will it occupy on a 32-bit architecture?**

1,000,000 nodes in ~20 levels per earlier estimations

4B per integer \* 1,000,000 integers = 4MB (not including overhead)

**My copy of Meyer’s Object Oriented Software Construction has about 1,200 body pages. Assuming no flow control or protocol overhead, about how long would it take to send it over an async 56k baud modem line?**

1200 pages \* 2KB/pg = 2400KB

56 Kb/s is really 7KB/s

2400 / 7 = mid-300s seconds or about 6 minutes

**My binary search algorithm takes about 4.5mS to search a 10,000 entry array, and about 6mS to search 100,000 elements. How long would I expect it to take to search 10,000,000 elements (assuming I have sufficient memory to prevent paging).**

10k is about 13 levels based on earlier estimations

13 levels / 3.5 mS is roughly 3 levels/second

100k is about 17 levels based on earlier estimations

17 levels / 3.5 mS is roughly 3 levels/second as well

10M is about 24 levels

24/3 = 8 mS

**Unix passwords are stored using a one-way hash function: the original string is converted to the ‘encrypted’ password string, which cannot be converted back to the original string. One way to attack the password file is to generate all possible cleartext passwords, applying the password hash to each in turn and checking to see if the result matches the password you’re trying to crack. If the hashes match, then the string you used to generate the hash is the original password (or at least, it’s as good as the original password as far as logging in is concerned). In our particular system, passwords can be up to 16 characters long, and there are 96 possible characters at each position. If it takes 1mS to generate the password hash, is this a viable approach to attacking a password?**

96^16 = roughly 10^16 = 10,000,000,000,000,000 ms to generate all hashes

Not really viable.