12.1

The primary problem is how do we store the data gathered via FTP so that we can do a large variety of analyses.

Method 1:

Store the data in text files. This would be easy to store but difficult to manage and update, as well as very hard to query. Keeping unorganized text files would lead to an inefficient data model.

Method 2:

Store the data in databases. Database stores data logically and provides an easy way of doing queries over the data. It also makes the maintenance easy and supports rolling back, backing up data and security using standard database features.

Method 3:

We can also store data with XML format if we just do a simple ayalysis and distribute the data. XML is a standard data model to share or distribute and there are efficient parsers are available to parse the data and extract out only the data that is desired. But it’s also hard to query.

12.2

We first think about the simple case that we can represent a person with his/her friends like this way:

class Person

{

Person[] friends;

//Other info

}

If we want to find connections between two people, we can start with one person and do a breadth first search.

But we can’t do this when there are millions of persons as it’s too large.

We can’t store all data on one machine as we have so many users information. We distribute all persons’ information into different machines then we replace the list of friends with a list of their IDs and traverse like this:

1. For each friend ID, find which machine it is on: int machineIndex = loopupMachineForID(id);

2. Go to machine whose id is machineIndex;

3. Look up the friend: Person friend = lookupFriend(id);

There are also many optimizations we can do:

1. Optimization: Reduce Machine Jumps

Jumping from one machine to another is expensive. Instead of randomly jumping from machine to machine with each friend, try to batch these jumps—e.g., if 5 of my friends live on one machine, I should look them up all at once.

2. Optimization: Smart Division of People and Machines

People are much more likely to be friends with people who live in the same country as them. Rather than randomly dividing people up across machines, try to divvy them up by country, city, state, etc. This will reduce the number of jumps.

The data structure we need in our algorithm is in java file “C12\_2.java”.

12.5

Let’s assume that the “same” page means the same URL. The web is a graph structure and we can traverse it using DFS or BFS. We can use a hash table to mark the visited pages just as we would use in DFS and BFS and the hash table would help us to get into infinite loop.

12.6

Pages are huge. We can use urls as representation of pages. We can use a hash table to check if an url has occurred before.

Then the algorithm is as follows:

1. Iterate through pages and compute the hash value of each one.

2. Check if the hash value is in the hash table. If it is, throw out the url as a duplicate. Otherwise, insert the value into hash table.

But memory limit is a big problem for this algorithm. Let’s assume each page hashes to a 4 bytes hash value and each url is an average of 30 characters, then each url will take up 34 bytes.

34 bytes \* 1 billion = 31.6 GB, so we could split the hash table across machines. Assume we have n machines, then the algorithm changes to follows:

1. Each page hashes to a hash value v.

2. v%n tells us which machine this page’s hash table can be found on.

3. v/n is the hash table that is located in its machine.

12.7

Construct an index for each field that requires range queries. Use a B+ tree to implement the index A B+ tree organizes sorted data for efficient insertion, retrieval and removal of records. Each record is identified by a key (for this problem, it is the field value).Since it is a dynamic, multilevel index, finding the beginning of the range depends only on the height of the tree, which is usually quite small. Record references are stored in the leaves, sorted by the key Additional records can be found by following a next block reference Records will be sequentially available until the key value reaches the maximum value specified in the query. Thus, runtimes will be dominated by the number of elements in a range.