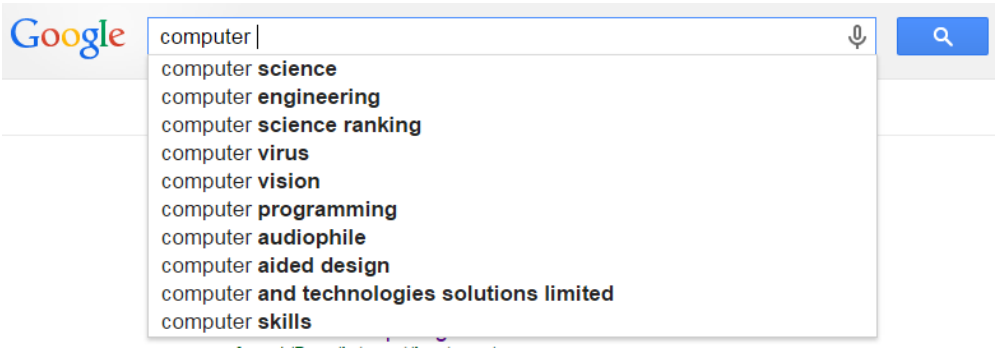
COMP 4021 Internet Computing

Homework 3

Due: May 10 (Sunday) 11:59pm

Total points (without bonus): 16 points; you can get up to 18 points with bonus.

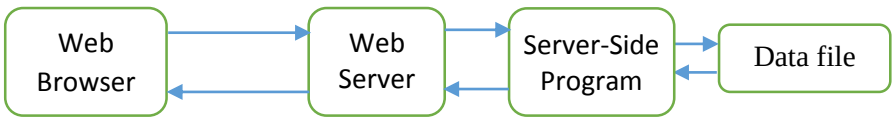
In Homework 2, you have already seen some JQuery codes for communicating between client and server. Since JQuery is by far the most popular JavaScript library, we will learn more about JQuery in Homework 3 by developing an application similar to the query autocomplete feature that is commonly used in search engines, e.g.,



In Homework 3, you are required to do the following:

1) Learn JQuery yourself since there will not be any labs or sample codes given for building this application.

2) The following is a data flow of the autocomplete requirement:



The data file can be downloaded from this link (or you can go directly to the website: http://www.countries-list.info/Download-List). You are required to do the JQuery programming on the browser side and the program on the server side. You can use any language to implement the server-side program. On the browser side, you use JQuery autocomplete widget.

3) Implement the following functional requirements:

i) When the user types in abc, you should list all strings in the data file that *contain* abc *anywhere* in the strings (e.g., . abc**d**…, **d**abc…, **d**abc**e**…).

ii) Show a maximum of 10 suggestions in the same order as they appear in the data file.

iii) The user can select any of the suggestions, and the suggestion will be sent to the server, and the server will return a "result page" which simply prints out the message "your query [the suggestion submitted] has been received".

iv) Allow 0.2 sec delay in between keystrokes before suggestions are updated, and trigger autocomplete only after more than one character have been inputted. E.g., user types in: 'a'

[delayed 0.5 sec]-> ‘b’ [delayed 0.3 sec] -> ‘c’ [delayed 0.1 sec] -> ‘d’ [delayed 0.3 sec] -> ‘e’ [delayed 0.4 sec] … Then, suggestions will not be shown after ‘a’, or ‘c’ but will be shown after ‘b’, ‘d’ and ‘e’.

v) As shown in the screenshot above, use regular font to display the matched substring and bold font for the remaining part of the suggestion. User types in ‘tab’, suggestions like ‘**da**tab**ase systems**’ will be shown. You can decide whether to write client-side code or server-side code (or both) to apply bold font to the suggestion string properly.

vi) Reference #1 below is a sample code which also implements client-side caching. The cache array saves previous suggestions, so if there is a hit, there is no need to query remote server. In this homework, caching is not needed although you can still implement it.

vii) Reference #2 below is a sample code that stores the suggestion data in an array. This code is for reference only. You are required to fetch suggestions from the server.

viii) You can design your own data structure and pattern matching method. For example, you can assume that the server program loads the data file into a main-memory data structure and then pattern matching can be conducted on it. It is up to you to choose the main-memory data structure to use.

ix) In addition to the client-side and server-side code, also upload a file containing a half-page description of your design.

x) Hint: You can assume that all suggestions (country names) in the data file can be loaded into main memory (e.g., a session array) at the beginning of a session, and then pattern matching can be done using PHP string match function by looping through the array. Note that this is only a hint, not a requirement.

xi) For anything unspecified here, you can make reasonable assumptions (e.g., how to handle capital letters).

Bonus: You can attempt the following bonus feature: [2 points] The server records every selection (click) of the suggestions, and maintains a count on the number of clicks on each suggestion. The server returns the matching suggestions in reverse order of the count.

You will need to do sorting on count data in PHP. You can assume that the count data is per-session based. That is, you can create a session variable (array, etc.) to store all suggestions when the session is started (as in requirement x above) and then access count data is updated in the array. You do not need to handle concurrent access and persistence of data. That is, if the server is down or the session is closed, it is fine that all suggestion and access data are lost.

Note: Speed does not matter for the basic and bonus requirements. As long as it works functionally, it is fine.

References:

1) https://jqueryui.com/autocomplete/#remote-with-cache (client side)

2) https://jqueryui.com/autocomplete/#default

Note: This homework is set by Prof. Lee. If you have any question, email to Prof. Lee (dlee@cse.ust.hk) but not the TAs