**Appaco\_Lib**

**Approximate autocomplete Library**

**Introduction**

Appaco\_Lib is library which facilitates and speeds up writing by offering a list of words (suggestions) that complement the few characters typed into the input text, while allowing errors in the prefix entered by the user, all by improving the quality of the results (static/dynamic) with ranking system, according to the importance (score) of words.

And it’s support static and dynamic ranking.

**Appaco\_lib features:**

* exact autocomplete : list of suggestion contain only words that begin by the same prefix typed in text box.
* approximate autocomplete : we allow error in prefix typed in text box, it's mean we show word that begin by the same prefix and also the word that begin by prefix similar to the prefix typed.
* static ranking : words will be displayed according to the importance of words.
* dynamic ranking : static ranking + update the score of the selected word (example +1)
* local support for dictionary source, it support huge number of words all in local (example English dictionary).
* Ajax query to get the results from server (or sometimes the dictionary source file)
* very fast response (trie based).
* sport one character typed to get results (use the ranking system)

**Supported language:**

appaco\_lib is coded in two language : javascript and c/c++.

**JS appaco\_lib :** used in client side, do all the work locally, and it can get result from server side.

**C appaco\_lib :** can be used in any type of application, and also we can use it in a server side using the module FastCGI.

those two library can be used separately.

and also we can use them both in the same context, js version in client side and c/c++ on server side.

**JS appaco\_lib :**

* *approximate\_autoComplete*

*(input\_id,source,options\_in);*

* *typeSearch*
* *typePrintResults*
* *isExactSearchMethod*
* *topK\_CHAR\_SEPARATOR*
* *words\_SEPARATOR*
* *doSort*
* *TOPK\_SIZE\_GROUP*
* *delay*
* *minLength*
* *sourceFromUrl*
* *resultsFromUrl*
* *score\_update\_unit*
* *item\_ID\_OnClick\_nextResult*
* *dataToSend*
* *ajaxRequestMethod*

**C appaco\_lib:**

* *Options\* new\_options();*
* *void print\_options(Options \*my\_option);*
* *void initialize\_AutoComplete*

*(const char \*source,Options \*my\_option\_in);*

* *ListString\* get\_results(char\* queryWord);*
* *ListString \*get\_next\_result();*
* *void Print\_ListString(ListString \*list);*
* *void json\_Print\_ListString(ListString\* list);*
* *typedef struct Options {...} Options;*

**Detailed Description:**

**Autocompletion:**

Is a technique that facilitates and speeds up writing by offering a list of words (suggestions) that complement the few characters typed into the input text.

It’s mean all the words (or a group of theme) in the dictionary (list of words given) that have the same prefix (begin of word) entered by the user in the text box.

when the user type 1 character the system provide all (or group of them, it’s depend on the system configuration) words that begin of this character

And then when he type the 2nd character, the system show words that begin with those 2 characters, the number of results decrease, and so on each time the user type another char the number of result decrease to become just a few words in the suggestion list, the user will find what he search or he will continue typing all his word till the end.

this technique is used in web browsers, search engine, smartphone, text editor…etc.

***image (autocomplete)***

**problem:**

in many time, the user when he type the few character of his words he may make an error

it’s maybe an keystroke error it’s always happen, or the user don’t know how to write this word …

in this case the result will not appear, or it will be appear but for the others words (words that have this prefix)

so in this case the user is not satisfied by the result.

**approximate autocomplete :**

an approximate autocomplete is exactly the same as autocomplete, unless in this case we allows the error at the prefix typed by the user, when the user types a few characters of a word (prefix), the system offers him a list of suggestions that contain all the words lexically similar to that prefix.

First, it suggests the keywords they have exactly the same prefix (exact autocomplete), then it suggests the keywords they have similar prefix (approximate autocomplete).

*for example : see this picture*

*we inter the prefix frinc, so google propose to use this result : france…etc.*

approximate autocomplete is based on number of error (k-error), if k=1 so in the prefix we accept 0 or 1 error. the number of error accepted is <=k

*image exemple (from google ) or better idea, put our system image example ☺*

**Ranking :**

ranking is classify the results according to certain metric in order to show the most important results in top of the list

when the system show us the suggestion , so the words in the list appears by alphabetic order (dictionary search) or it’s maybe the number of searching this words by all user like goggle serach engine.

so there’s a metric how the result appear and how they ordered.

Example :

System that search all cities, and the results appear depending on the score (importance) given to each city.

[……., ”paris usa#50”,“paris france#100”,……. , ” paris yyyy #65”]

When th user type : paris

result : paris France , paris yyyy , paris usa…..,

paris France is the first because it have a score greater than others cities.

**Static/dynamic ranking:**

the score can be static (ie not editable), or dynamic (update the score of the selected word (example +1)

example

static ranking : words cities score, alphabetic order dictionary…

dynamic ranking: google suggestion (or web page search).

**appaco\_lib in details :**

Most of the efficient approximate prefix searching algorithms first build an type of index on the dictionary (list of words) and then use this index to perform each query efficiently.

In appaco\_lib We build an index also, that can be compared to a ranked trie of all dictionary words.

Then this trie is used to answer queries in an approximate way, using a ranked traversal of some promising nodes of the trie.

This library (client side) my combined with the server side (using AJAX) to get results (or dictionary source file) depending on the best scenario.

For suggestion results : we do all the work on local for the small dictionary as English or French dictionary.

But if we have a very big dictionary (or database) so here we can specify in options that the results come from server.

For the server side: we have also the same library written on C/C++ and it’s running on fascCGI,

Of course, the developer can specify the server he wants for example php or other ...

**How to use the JavaScript Appaco\_lib:**

We begin by very simple explanation and then we detail the entire library.

two file to include in your project :

1. approximate\_autoComplete.js
2. approximate\_autoComplete.css

And then call the follow function in your JS program :

*approximate\_autoComplete(input\_id,source,options\_in)*;

**when**

**input\_id:** is the id of the text field in your HTML page.

Example :

<input id="search" type="text"/>

So input\_id = "search";

**source:** the source for building the dictionary that allows to make the approximate autocomplete.

it can be an array, string, URL, array object ...etc

we will return in detail about it by the following.

Exemple :

Source = “abcd,abcdefg,abxye,…., zzz”;

Source = [“abcd”,”abcdefg”,”abxye”,…., “zzz”];

**options\_in :**

the different options to build the dictionary and make the search on it and display the result.

if we want the system takes the default options, in this case we can proceed without specifying this parameter.

but it’s very important to specify some of options if we want to personalize your system

We have many options, and all of them are in one object (literal object) we specify the option name and its value, we can modify just some of options or all of them.

It’s obvious that we separate the options by ‘,’ cause we use a literal object.

Example (just three options):

options\_in = { words\_SEPARATOR :’\n’, minLength :3, doSort :true};

We will explain all options one by one in detail later see (link to explanation all options :p )

**Example the use of 'JS Appaco\_lib':**

Include the css and JS file:

<link rel="stylesheet" href="approximate\_autoComplete.css"/>

<script type="text/javascript" src="approximate\_autoComplete.js"> </script>

Html input text:

<input id="search" type="text"/>

**example 1:** in this case we don’t specify the options (we take default options)

Var source = [“abcd”,”abcdefg”,”abxye”,…., “zzz”];

approximate\_autoComplete('search',source);

**example 2:**

var source = “abcd,abcdefg,abxye,…., zzz”;

approximate\_autoComplete('search',source, { words\_SEPARATOR :’,’, minLength :2});

**Source:**

A list of words (and their score) to build the index (dictionary) that allows making the approximate autocomplete.

The words score is optionally, we puts it if we want to do a ranking search.

We have different type of source, to make developer feel free in their choice depending on the situation needed.

**Source from local:**

The source of words is in local, the construction of index and search is in local.

1. **String :** the source is a string, we have to indicate the words separator ‘words\_SEPARATOR:value’
2. **A simple string :**

We use this technique generally if we have some of words that we can put them manually (of course we can get them automatically …).

source ="abced,efg,abcdxx,abcdrohhj,ibraa,ibororo,.......,zzzz";

option\_in = { words\_SEPARATOR :’,’};

for words scores, it's simple, just follow the word with his score and separate them with a special character ‘topK\_CHAR\_SEPARATOR’, ofcourse it’s must be deferent from words\_SEPARATOR.

source ="abced#10,efg#0,abcdxx,abcdrohhj#9,ibraa,ibororo,.......,zzzz";

option\_in = { words\_SEPARATOR :’,’, topK\_CHAR\_SEPARATOR :’#’};

we can put the score just for some words

(as we can put it for all words).

In the case that we don’t specify the score, the default value is '0'.

1. **string from file (big dictionary as english.txt)**

this is the same of the previous one, just here we have to read the local file (for example HTML5 File API see ‘<http://www.w3.org/TR/file-upload/>’).

We use this method when we need to use a lot of words for example a real dictionary as English or French…etc.

Remark:

* Generally in dictionary we have each word in a new line, so in this case we can let ‘words\_SEPARATOR’ by default cause

The default value is: words\_SEPARATOR:/[\n\r\t\0]+/g;

* we must be very careful with ‘words\_SEPARATOR’ in file, cause it’s depend on witch OS you use, for example in windows we use \r\n to return to new line, but in linux we use just ‘\n’, and on mac ‘\r’

see this : <http://en.wikipedia.org/wiki/Newline>

if it is certain that each word is a new line (a word or a word with his score), we can leave the default value of ‘words\_SEPARATOR’

example: get all words from a local dictionary

var file\_input\_id = 'file';

var fileInput = document.querySelector('#'+file\_input\_id);

fileInput.onchange = function()

{

var reader = new FileReader();

reader.onload = function()

{ // source == reader.result

approximate\_autoComplete('search',reader.result,{ topK\_CHAR\_SEPARATOR:"#",minLength:1});

};

reader.readAsText(fileInput.files[0],"UTF-8");

};

* the file must encoded in UTF8 or UTF8 without bom (it’s better).

1. **array :**
2. a simple array of string:

each element is a string that contain a word (and its score)

source = ["abced","abcdefg","xyuss" ,.....,"zzzz"]

or

source = ["abced#10","abcdefg","xyuss#23",.....,"zzzz#01"]

in the second example we specify the score separator {topK\_CHAR\_SEPARATOR:"#"}

1. An array of an literal object as follow :

[{label,value},{label,value},{label,value},....,{label,value}]

Label : for the word (string)

Value : for the score (integer)

this method is just to separate and organize the data, for example we can extract information from an XML file and then put them in this type of array. Also we can get this type when we use ‘Json’….

source = [{label:"abced",value:10},......,{label:"zzzz",value:0}]

1. An array of array as follow

[[],[],......,[]]

each internal array contain a word and its score separated by ‘,’

We use this technique to simplify the previous method (array of literal object)

here we haven’t to specify a label and value each time.

Example :

source = [["abced",10],......,["zzzz",0]];

**Source from server :**

the source to construct the index is in a server, the file is in the same server of the web page or on distant one.

for example if you have a dictionary file you can put it in your server and use it (example : English.txt)

**How to use it:?**

1. we put the file path (absolute path or relative) in source variable.
2. and we put the option 'sourceFromUrl' to true, {sourceFromUrl:true}

**example:**

source = ‘www.example.com/english.txt’;

or

source = ‘english.txt’; this if we are in the same folder at the same server.

Option\_in = { sourceFromUrl:true , doSort:true};

approximate\_autoComplete('search',source, Option\_in);

**Remark:**

1. the dictionary must be lexicography ordered, so we put the option ‘doSort’ at true to sort all file words, but we know that this operation (sorting) take a lot of time, in fact, it takes most time in the process of building the index.

to avoid this, it’s better to sort all the words and save the file for once, and then you disable this option {doSort:false}, so the construction will be very fast.

1. If your file is in deferent origin ( for example : deferent domain name), so you must be very carful, because there is a security concept for a number of browser-side programming languages, there is an access restrictions

You can see: <http://en.wikipedia.org/wiki/Same-origin_policy>

we must be sure that the other site (domain) allow cross-domain request.

1. Zip /Gzip file to spead up the file transfer: you can compress your file and configure your server to deal with zip file, the browser will unzip automatically the file.

**Results:**

The results is displayed automatically in a pop up menu linked to textbox.

The results come from tow source local or from server.

The number of words that will be displayed at the same time in the suggestion list (pop up menu) is depend on the Tow options ‘typePrintResults’ and ‘TOPK\_SIZE\_GROUP’

The first option defines how we display the result group by group or all results at once and the second define the number of words displayed at the same group.

Go to the section options to see more detail about those tow option.

**Results from local:**

All work is done in local (client side), the construction of the index, and the search of the queryword, and then we display the list of the suggestion.

In this case we must provide a list of words to construct the index.

**Results from server:**

All work is done on the server, so we use an AJAX request to get result from server and then displayed it.

We have to specify the source of the result (the server address)

and we haven’t to create an index in a local.

we use this if for example we have a very big dictionary or database on a server.

How to use it :

1. We put the option ‘**resultsFromUrl**’ at **true**
2. and we specify the URL of the server in source

Source : [www.example.com](http://www.example.com),

the server can be php or others, we can use our library ‘c appaco\_lib ‘ in a server side to deal with very big dictionary, (it’s run under FastCGI).

Example :

Result come from server php :

var source="approximate\_autoComplete.php";

approximate\_autoComplete('search',source,{ resultsFromUrl:true});

*See example (mettre les 2 lien, celle de php et celle de c appaco\_lib fastcgi)*

**next results**

when we display results by group, we have an option ‘next\_result\_button’ to display all results group by group, (of course if he didn’t find what he search)

to get next result (next group of words) generally the user click on a link (for example google page).

In our case its simple, it suffices to indicate the identifier ‘id’ of the html item (link, button) to the option ‘next\_result\_button’. (it’s work on the event on click)

Example

/---------------------------------------------------------------------------------------\

<input id="search" type="text"/>

<input type="button" id="next\_result\_button" value="next">

<script>

Var My\_option={item\_ID\_OnClick\_nextResult:'next\_result\_button',score\_update\_unit:0};

approximate\_autoComplete('search',source, My\_option);

</script>

\---------------------------------------------------------------------------------------/

And also he can use CTRL+RIGHT to do the same thing.

When we use this technique we don’t search the query word each time, we just go to our index and choose the K next element to display.

The k next element is not pre-saved because it take a lot of memory space, and also it take time, so we do this on the demand of the user, generally the user need just the first group of result and not all the results.

**Remark:**

1. If the user don’t wants to go to next result and his word doesn’t appear in the list so he just types some of other characters of his words to do more filtering.

**score and ranking**

to do a ranking search we give a score to all/some words to describe a importance degree of.

Example:

Source = ["abcd#10" , "abcdefg#5" , "xyz#02" ]

The number of words that will be displayed at the same time in the same group according to (top-k : score of the words in dictionary), so choose the top –k element according to their score, if there is no score, so the order will be according the alphabetic order.

**Static and dynamic ranking:**

This to describe if the score initial given to each word change or not, so We can update the score of words, we have an option ‘**score\_update\_unit**’, we give it a integer value as follow

0 : it’s mean we will have a static ranking, no update, the score initial remain by default.

Other unit : dynamic ranking, it’s can be a positive/ negative value, it’s mean the initial score will be changed with this unit. (if we have a positive value so the score will go up, example the google system +1)

**Options:**

In this section we going to explain all type of option

We have 16 options, some of them are very important like ‘doSort’, and other less important, some of them you can let their default value, and other you have to change theme.

**typeSearch :**

search types, how to search query word in the trie, we have three way

1. search query word each time from root

(SEARCH\_WHOLE\_WORD\_EACH\_TIME or 0 )

1. search the second query word from the node of the first query word, if the first is total prefix of the second, So it comes to search just the suffix added

(SEARCH\_SUFFIX\_BY\_PREFIXE or 1 )

1. we start from the node where the two query word share a prefix, in the 2nd query we delete or add character, and then the serach begin from the node of the same prefix with the 1st queryword.

(SEARCH\_DELET\_SUFFIX\_ALL or 2 )

The default value is : SEARCH\_WHOLE\_WORD\_EACH\_TIME

**typePrintResults:**

display types, how to display the result

1. simple display, words are in the list in alphabetic order

(PRINT\_WORD\_LIST or 1 )

1. words are ordered according to the top k (words score), and they are displayed all at once

(PRINT\_WORD\_LIST\_TOP\_K or 2 )

1. words are ordered according to the top k (words score), and they are displayed group by group **(with "next" option)**

(PRINT\_WORD\_LIST\_TOP\_K\_byGroup or 3)

The default value is : PRINT\_WORD\_LIST\_TOP\_K\_byGroup

**isExactSearchMethod :**

exact or approximate search with one error. (exact/approximate autocomplete)

1. exact search : isExactSearchMethod = true
2. approximate search : isExactSearchMethod = false

The default value is: false (approximate autocomplete)

**topK\_CHAR\_SEPARATOR  :**

a character separator between a word and it’s score

example topK\_CHAR\_SEPARATOR = '#'

["abcd#10" , "abcdefg#5" , "xyz#02" ]

The default value is : ‘#’

**words\_SEPARATOR :**

the separator between words in source (if it’s string of course)

it can be regular expression or a simple string or character.

example words\_SEPARATOR=','

source = "abcd,abcdefg,xyz,......,zzzabc" ;

The default value is : /[\n\r\t\0]+/g (regular expression to deal with words in file separated by a new line)

**doSort:**

sort the dictionary or not, (before the construction of the Index)

so {doSort:true} sort the dictionary in lexicographic order

this option is very important one, so we have to deal with it carefully,

sorting a dictionary take a lot of time,

we disable this option {doSort:false} if and only if the dictionary is sorted.

otherwise we may not have good results, because the index is poorly constructed.

The default value is : doSort = true;

**TOPK\_SIZE\_GROUP:**

The number of words that will be displayed at the same time in the same group according to (top-k : score of the words in dictionary)

example TOPK\_SIZE\_GROUP=10,

the list of suggestion will contain the TOP-10 words.

The default value is : TOPK\_SIZE\_GROUP=8,

**Delay:**

The delay in milliseconds between when a keystroke occurs and when a search is performed. (the waiting time befor we bagin searching the queryword in the trie)

if the results come from a server, do not put **Delay** to small values ​​because it may overload the network.

Generally ‘delay’ is put at 300 milliseconds.

The default value is : delay=300

**minLength:**

The minimum number of characters a user must type before a search is performed.

example minLength=3 , research will not begin until the 3rd character.

**Remark:**

* If we put **minLength** at 0 or 1, it’s mean we will have all the dictionary as results, cause we will have all words that begin with this unique char and also all the other approximately (with one error).

So if we display the result by group here it’s will work very well

But if we try to display all the result at the same time, here it’s a big problem; the browser will crash.

* In jQuery they say “( Zero is useful for local data with just a few items)”

In our lib we haven’t this problem because we use an index that allows us to speed up the search and then we display results by group.

**sourceFromUrl :**

Here the source for the construction of the index is in a server, the file is in the same server or on another server.

sourceFromUrl= true.

Default value : sourceFromUrl:false

For more detail see ‘source from server’.

**resultsFromUrl :**

specify the source of the result, (locale or server)

if we put the **resultsFromUrl = true ,** it’s mean that the result come from server,

For more detail see ‘results from server’.

Default value : **resultsFromUrl = false.**

**score\_update\_unit:**

the update unite for the words score

0 : a static ranking, no update.

Other unit : dynamic ranking.

Default value : **score\_update\_unit = 0**

**item\_ID\_OnClick\_nextResult:**

The id of an item (button, link) to use to get Next group of result when the event on click happen.

For more detail see ‘next results’ section.

Default value item\_ID\_OnClick\_nextResult: 'undefined'

**dataToSend:**

we use this if the result come from server ‘resultsFromUrl=true’, so we specify all types of data (name/value) you want to send to a server, of course the queryWord is automatically added.

Default value : dataToSend:{SIZE\_GROUP :10,minLength:2,delay:300}.

**ajaxRequestMethod:**

specify the method to use (get or post) to send data to server when the results come from server, 'resultsFromUrl=true'

default value : ajaxRequestMethod = GET;

**Keyboard interaction (ibra : changer la place de cette section)**

When the menu is open, the following key commands are available:

* UP : Move focus to the previous item. If on first item, move focus to the input. If on the input, move focus to last item.
* DOWN : Move focus to the next item. If on last item, move focus to the input. If on the input, move focus to the first item.
* ESCAPE : Close the menu.
* ENTER : Select the currently focused item and close the menu.
* CTRL+right : display the next group of results.

**C Appaco\_lib:**

the approximate autocomplete library "appaco\_lib" coded in C language.

all the files is in a folder named "appaco\_lib"

the interface of the library is in the file "approximate\_autocomplete.h"

the interface :

* Options\* new\_options();
* void print\_options(Options \*my\_option);
* void initialize\_AutoComplete(const char \*source,Options \*my\_option\_in);
* ListString\* get\_results(char\* queryWord);
* ListString \*get\_next\_result();

Explanation:

**Options\* new\_options():**

default configuration for research (exact or approximate, how to search, how to display the result)

that exactly the same of option in JavaScript library.

this function create a new structure who contain all options needed for the creation and working of "approximate autocomplete".

**void print\_options(Options \*my\_option):**

this function display the state structure 'Options' to see the different options (variable) and their value.

**void initialize\_AutoComplete(const char \*source,Options \*my\_option\_in):**

build the index (ranking Trie) from an input file that contains the dictionary.

it takes as parameter:

1. const char \*source : we put the file path (absolute path or relative) in source variable., this file contain all our dictionary (list of words and their score).

**Remark:**

in c appaco\_lib version , we use a single source, a file that contain the words separated by a new line.

* the file must encoded in UTF8 or UTF8 without bom (it’s better).

1. Options \*my\_option\_in : data structure that contain all options needed for the creation and working of "approximate autocomplete".

with "new\_options()" we creat a default options, and then we can change some/all options in this structure (example: my\_option\_in->data = value;)

**ListString\* get\_results(char\* queryWord):**

search the query word and return results (a list of words)

this function is very simple, we give her a queryword and it return a list of suggestion

of course before we call this function we have to create index by (initialize\_AutoComplete)

parameter :

**char\* queryWord :** a string, a few characters typed by the user in text box.

Results:

**ListString :** a structure that contain results:

Typedef struct ListString

{

char\*\* buffer;

int capacity;

int size;

} ListString;

* char\*\* buffer : an array of string that contain words
* int size : number of words in the array (buffer)
* int capacity : total capacity (total number of cell) of our array (buffer)

**remark :**

capacity >= size

so we use simply a loop to get all words

example :

\--------------------------------------------------------------------------------------------/

int i;

for(i=0; i<my\_result->size;i++)

{

printf("%d) %s",i,my\_result->buffer[i]);

}

\--------------------------------------------------------------------------------------------/

**Remark :**

You can use 2 very simple function (available in c appaco\_lib)

* **void Print\_ListString(ListString \*list) :** print a list in a console.
* **void json\_Print\_ListString(ListString\* list) :** print an array in json forma (with "Content-Type: application/json; charset=utf-8\n\n" comme header)

use this function if you use ‘c appaco\_lib’ in server side (CGI/FastCGI)

**ListString \*get\_next\_result():**

Return the next group of results (of course if the system is configured to display result by group), this function must be used after the function ‘get\_result’.

example : console apps/ server side

/-----------------------------------------------------------------------------------\

#include <stdio.h>

#include "appaco\_lib/approximate\_autocomplete.h"

int main()

{

char source[100];

char queryWord[100];

// 1st phase : initialisation

printf(" \n\n enter the dictionary name (path) : ");

gets(source);

Options \*my\_option = new\_options();

// you can change some/all option after this instruction.

my\_option-> minLength = 3;

my\_option->words\_SEPARATOR = ‘\n’;

initialize\_AutoComplete(source,my\_option);

// 2en phase : approximate autocomplete

printf(" \n\n enter the query word : ");

gets(queryWord);

// in the server side, we have to get this value from the global variable environment/console (it’s depend on how the data was sent get/poste)

ListString\* choices=get\_results(queryWord);

Print\_ListString(choices); // display a list of suggestion

// in the server side we use json\_print\_ListString(choices);

return 0;

}

\--------------------------------------------------------------------------------------/

All options in c appacol\_lib:

All the options are the same as javascript lib, if you need explanation go to lien.

typedef struct Options

{

int typeSearch;

int typePrintResults;

bool isExactSearchMethod;

char topK\_CHAR\_SEPARATOR;

char words\_SEPARATOR;

bool doSort;

int TOPK\_SIZE\_GROUP;

int delay;

int minLength;

} Options;

As we can see there’s some of option don’t exist in ‘c appaco\_lib’, because we don’t need them, but in JavaScript we need them to deal with result come from server.