Words Autocompleting Using Nesting Matrices

Elaf Sabah Abbas*,M.Sc.(Asst. Lecturer)

Abstract: Since 1980's, word-prediction systems have been used as writing aids. They were used by people with physical incompetence to reduce the amount of effort needed to enter text, but later they were found to be also helpful to people with learning or language weakness, such as difficulties in speaking, spelling, or grammar. The purpose of this paper is to propose an auto-completing system built using multidimensional nesting matrices to build a database that store the previously learned words, and use that database to find all the words that starts with a given two letters. Where, it's taken 5.343 seconds to look for word which starts with existing first two letters, and it has taken 2.2568 seconds to look for word which start with two nonexistent letters in the database.

Keywords*:* Web browser, query auto-completion, autosuggest, Interface design.

* Al-Mansour University College, Communication Eng. Department, Baghdad, Iraq

1. Introduction

The storing of activity traces has been assistant the re-visitation of the page which can be considered as a convenience features for the Web browsers. These features has been realized a variety of context [1]. The set of suggestions that has been displayed to the user for the purpose of rounding what is looking for; this is the goal of the auto-completion. The small sufficiently selection has been conveyed to the use, when the group of potential suggestions is too large [2].

The auto-completion provided by many web browsers, e-mail programs, search engine interfaces, source code editors, database query tools, word processors, and command line interpreters [3]. For these kinds of applications, can easily achieve fast response times by two binary or B-tree searches in the (pre)sorted list of candidate strings [4]. Auto-completing is usually a services provided to the users during their search. Each time the user enter a character to the search box the auto-complete mechanism suggested the most matching candidates based on the entered characters. First and based on the prefix matching a filtering operation is performed by using data structure mechanism, and then based on the expected probability of the suggestions that match the prefix, a fetch operation is performed. The probability values are calculated approximately based on the past frequencies [5].

Autocomplete has been used by the programmers to reduce the task of remembering tiring lists of APIs. Autocomplete has a point of failure: when a programmer expects a certain method or function name to exist, and it does not, the auto-completion list simply stops working [6]. To reduce the amount of user input, the AutoComplete is a feature that provides a list of suggested items that closely match what the user has typed most often or recently. It speeds up human-computer interactions in applications and reduces the number of keystrokes. The AutoComplete is only applied when there are text fields and the word being typed can be predicted [7].

2. Related Work

Several methods are used in various application for the propose of auto-completion. Query auto-completion (QAC) is an amazing feature in the modern search engines. Each time the user entered a new character into the search box the query elects list is updated. Queries prefixes usually tend to be short and vague, and the existing models, mostly rely

on the past matching candidates for ranking [5]. Another method is the Keyword programming which is a new technique that reduces the need to remember all the details of syntax in the programming language and APIs, by interprets a group of keywords provided by the user into a valid expression. The keywords act as a query that searches the space of expressions that are valid in a given context. Prior work has demonstrated the feasibility and merit of this approach in limited domains [8]. Several studies investigating the auto-compilation, [1] have used a mixed methodology approach to study the occasional information that found in web browsers. Then, [2] was identified a number of key design dimensions of auto-completion interface components. It presents an entirely configurable architecture, which can be used to configure auto-completion components to the desired point in the design space. In the context, the contributions of the [9] are a classification schema for CTS approaches, identification of their deficiencies and then discussion of several important features. Whilst, [6] describes automatic function definition (AFD), which can succeed where autocomplete fails. A preprocessing, graph mining, and hashing for generating the suggestion list is proposed by [10]. While, [5] presents a supervised framework for personalizing auto-completion ranking. Test results show that the proposed method significantly outperforms existing document specific autocomplete search techniques.

3. The Proposed Auto-completion Mechanism

The proposed system is contains two stages, the training stage and a testing stage. In the training stage, a database of words is entered into the auto-completion algorithm and the following steps will be performed for each entered new word:

Step 1: Convert each word into its equivalent code that represents the position of each letter in the alphabet. (Like; a=1, b=2 and so on).

Step 2: Create a multidimensional array named “Link” of size (M, M), where M is the maximum number of possible links based on the used alphabet (26- for English alphabet letters).

Step 3: Check all the words of the training database to find the value of the third dimension of the “Link” array, which represents the maximum number of links found in the database. To find the number of links in each word of the training database is checked and every two consecutive letters are connected by a link. The maximum number of links represents the

longest word in the training database-1. (Example on that is shown in figure 1)

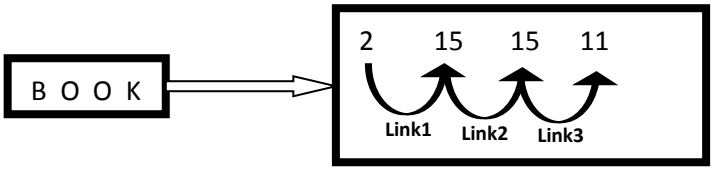


Figure (1): Coding and Links of word "Book."

The indices of the "Link" array are taken from each coded word as shown in figure 2.

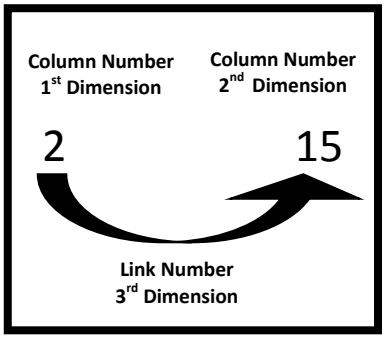


Figure (2): Indices of "Link" array in the proposed system based on "Book" word.

By performing these three steps on the training database, a multidimensional matrix is created which contains the sequence of the words used in the training database. As shown in figure 3.

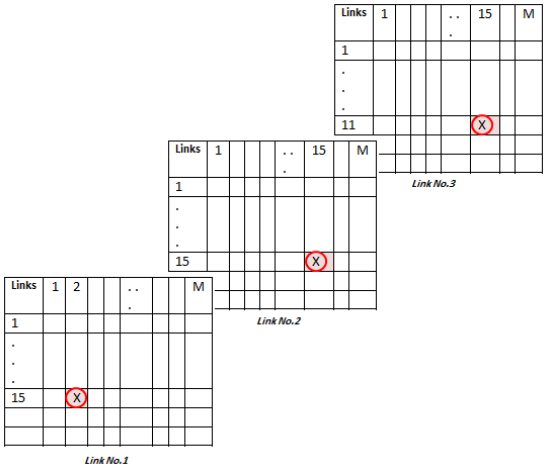


Figure (3): The Creation of the Suggested System matrix based on the indices taken from the letters numbers of "Book" word.

The overall design is shown in figure 4, where N (longest world letters-1 (45-1=44 links [11]) is the maximum number of links found in the training words database.

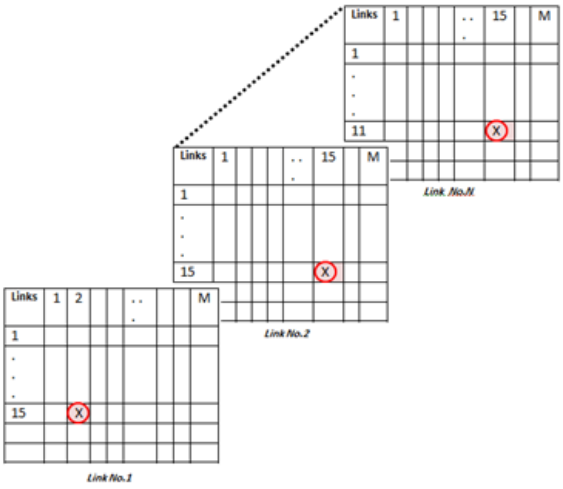
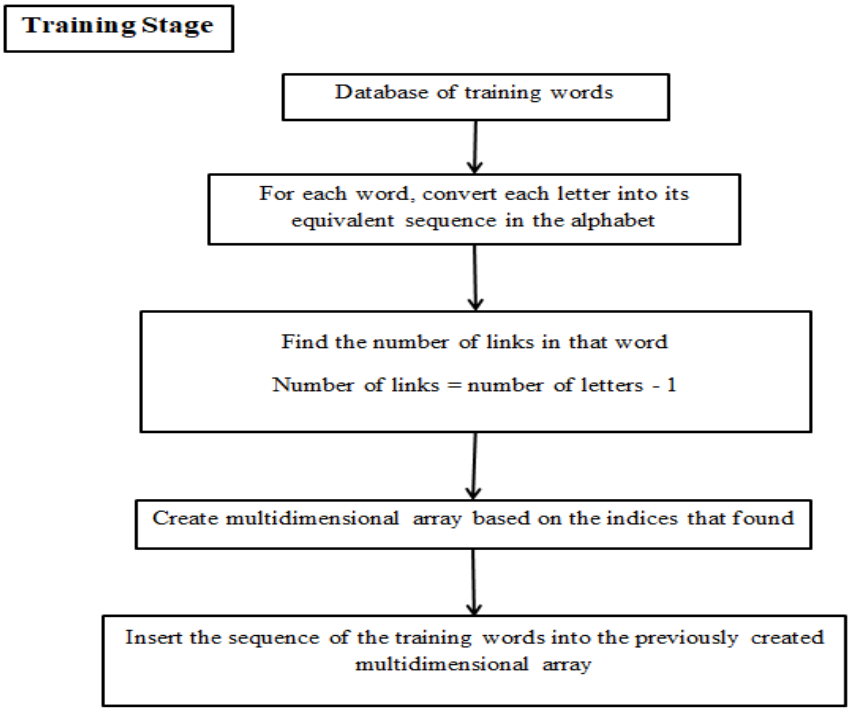


Figure (4): overall design architecture of the proposed system

In the proposed system it is necessary to perform the training stage before starting using the actual system, and the time required for the training stage is depending on the number of the words that are used (increasing the number of words cause increasing the time of the training), and hence the training stage is performed before the actual usage of the system, and the training stage is performed offline, so the time required for it is irrelevant. The most important time is the time required to find the word that's its two letters is inserted by the user either it's existed in the database or not.

A flow chart for the proposed system is shown in the figure 5.



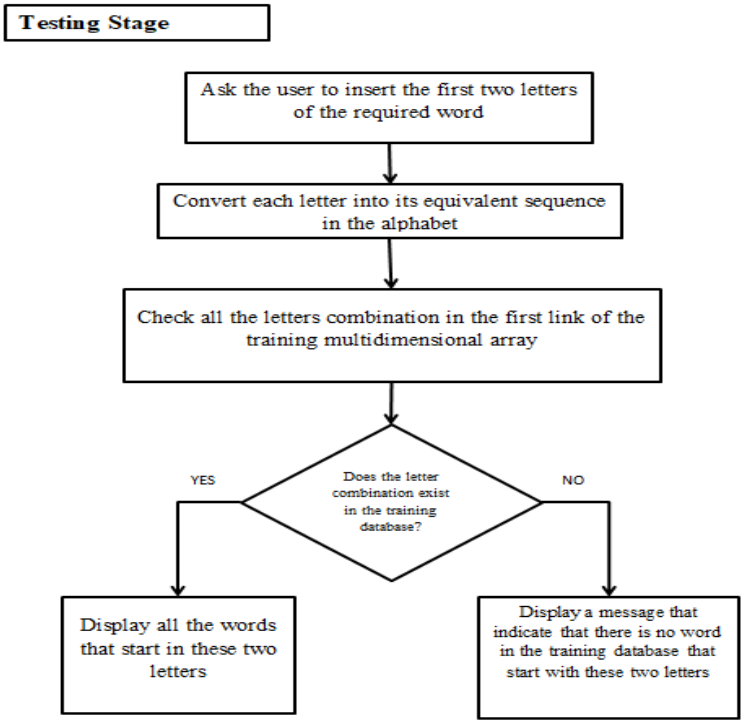


Figure (5): a flowchart of the proposed system that includes the training and testing stages.

4. The Proposed System Results

The proposed system has been built with 2000 words as database for training stage. The system required 5.343 seconds to search the data base for existing first two letters, and required 2.2568 seconds to search the database for nonexistent first two letters. It is worth mentioning that the time required to search the database for nonexistent first two letters is less than the time required to search for existing first two letters because the system doesn't search the entire database, its search only the first link of all array which taking significantly less time. Figure (6), shows the relationship between the number of words in a database and search time for all combinations with a starting two letters, as shown in the figure increase the number of words cause the increase in time required to search for all the combinations of the first two letters. Table (1) shows the required time to search for the combination of any two letters in the testing stage.

5. Conclusion

By running the program for all possible first two letters in the alphabet, the following concluded can be seen; The suggested system is the simplest type of the word auto-completing, and the main purpose of the system is to use multidimensional nested matrices. Also, the time required to find the words start with any two letters is directly proportional with the number of words in the training database. Whilst the results could be changes according to the database size and speed of operation processor. The concept of multidimentional nested matrices and using a previously trained system can be used for many classification applications, including linking diseases to a spesific symptoms and other systems that required linking information to get a specific results.

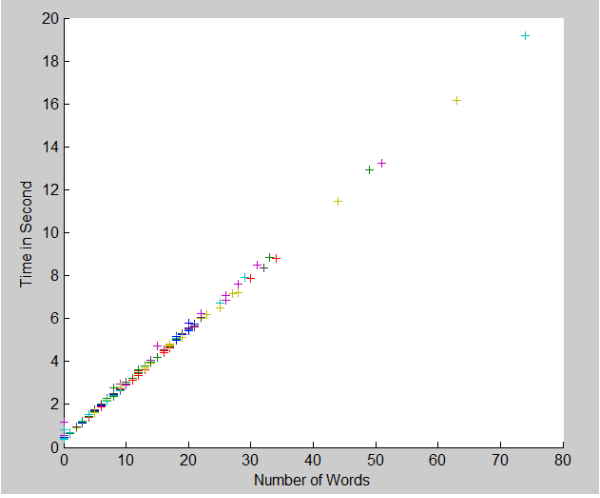


Figure (6): The relation between the number of words and search time for all combination with two start letters.

Table 1: Search time/number for each word

	a	b	c	d	e	f	g	h	i	j	k	l
a	0.4727	2.5813	3.7882	3.9299	0.4511	2.4689	3.6778	0.9403	1.2781	0.4181	0.5338	4.6866
b	6.7281	0.4759	0.4243	0.4144	8.3328	0.4262	0.4085	0.4057	2.4523	0.4091	0.4085	3.0219
c	6.8706	0.4224	0.4116	0.4159	2.0125	0.4179	0.5993	5.7698	1.7873	0.4060	0.4083	4.5093
d	3.3334	0.4129	0.4175	0.4168	12.4768	0.4353	0.4306	0.4101	9.8026	0.4462	0.4528	0.4424
e	4.6190	0.5245	0.5555	1.9471	0.4753	2.0196	0.7716	0.4684	0.7513	0.4960	0.4656	2.3344
f	8.2912	0.5324	0.5443	0.4730	3.6760	0.4939	0.5275	0.5091	5.1834	0.4421	0.5194	3.9843
g	3.1596	0.4953	0.6517	0.4683	2.1404	0.4863	0.4490	0.4605	1.3525	0.4567	0.4791	1.5435
h	6.2783	0.4278	0.4532	0.4910	4.7116	0.4283	0.4218	0.4345	2.4368	0.4763	0.4423	0.4416
i	0.4317	0.4400	0.7555	1.3393	0.4677	0.7205	0.4339	0.4347	0.4572	0.4289	0.4223	0.7357
j	0.7407	0.4317	0.4358	0.4317	0.4630	0.6108	0.5052	0.6580	0.6495	0.4695	0.5015	0.4629
k	0.4370	0.4318	0.4239	0.7256	1.0029	0.4415	0.4412	0.4288	2.2781	0.7499	0.4253	0.4371
l	4.8118	0.4267	0.4315	0.4277	5.1234	0.4356	0.4286	0.4371	5.9766	0.4203	0.4191	0.4156
m	7.4573	0.4944	0.4176	0.4448	6.0231	0.4524	0.4857	0.4807	6.0357	0.4444	0.4457	0.4284
n	2.1280	0.4192	0.6548	0.4234	5.8862	0.4303	0.4808	0.5455	1.1506	0.4983	0.4715	0.4161
o	0.4179	1.4517	0.9759	0.4558	0.4345	2.3151	0.4314	0.4435	0.7028	0.4240	0.4318	0.7309
p	8.1603	0.5336	0.4779	0.4793	5.6395	0.4254	0.4456	0.7927	2.9243	0.6828	0.5655	3.9691
q	0.4696	0.4503	0.5333	0.5325	0.4619	0.4478	0.4365	0.4367	0.4837	0.4500	0.4606	0.4392
r	4.0216	0.4567	0.4624	0.4305	17.1296	0.4411	0.4434	0.4206	3.6024	0.4467	0.4375	0.4271
s	6.2471	0.4670	4.0911	0.4626	8.2366	0.4560	0.4548	8.8874	5.5878	0.4417	1.4786	2.4616
t	2.7470	0.4368	0.4535	0.4273	4.3998	0.4711	0.4301	7.6586	2.7533	0.4691	0.4781	0.4899
u	0.4429	0.4517	0.5188	0.4329	0.4711	0.4398	0.7242	0.4546	0.4476	0.4540	0.4406	0.4324
v	1.7435	0.4723	0.4286	0.4299	1.6582	0.4387	0.4285	0.4278	2.4516	0.5079	0.4211	0.4554
w	5.2110	0.4540	0.4210	0.4549	4.7997	0.4503	0.4589	4.7562	5.8374	0.4458	0.4449	0.4346
x	0.4523	0.4463	0.4661	0.4306	0.4288	0.4218	0.4479	0.4274	0.4619	0.4331	0.4395	0.4563
y	0.7336	0.4477	0.4751	0.4410	1.7905	0.4261	0.4175	0.4493	0.6699	0.4441	0.4406	0.4647
z	0.4417	0.4405	0.4480	0.4524	0.7419	0.4612	0.4685	0.4576	0.4354	0.4418	0.4774	0.4487

m	n	o	p	q	r	s	t	u	v	w	x
1.9147	5.2483	0.4159	2.8087	0.4488	3.8913	2.8017	2.7643	1.2853	1.5877	1.4646	0.6831
0.4089	0.4106	6.4271	0.4130	0.4047	5.4740	0.4160	0.4115	4.3955	0.4113	0.5018	0.4981
0.4094	0.4313	24.4073	0.4158	0.4114	4.5608	0.4546	0.4692	4.4822	0.4206	0.4310	0.4137
0.4542	0.5316	4.6348	0.6484	0.6322	5.0502	0.9207	0.9376	2.9821	0.5283	0.5166	0.4971
1.6880	4.6303	0.4264	0.4326	0.7069	0.4284	1.5933	0.4241	0.4258	2.9916	0.4323	10.0369
0.4406	0.4655	7.0633	0.4846	0.4628	4.8447	0.5737	0.5843	4.6084	0.5385	0.5665	0.5417
0.4579	0.4205	2.2748	0.4545	0.4621	5.4223	0.4547	0.4570	1.9885	0.4305	0.4310	0.4304
0.4291	0.4721	5.5426	0.4368	0.4221	0.4289	0.4670	0.4406	2.7500	0.4674	0.4442	0.4355
3.0092	9.1192	0.4765	0.4217	0.4355	0.7335	0.7230	0.6844	0.4223	0.4361	0.4717	0.4355
0.4345	0.6027	2.0272	0.4358	0.4203	0.4302	0.4282	0.4266	1.9425	0.5576	0.5061	0.4289
0.4394	2.0280	0.4351	0.4296	0.4313	0.4305	0.4336	0.4348	0.4259	0.4326	0.5055	0.4501
0.4225	0.4455	6.2972	0.4243	0.4230	0.4297	0.4270	0.4124	1.4794	0.4279	0.4501	0.4341
0.4495	0.4290	6.0287	0.5083	0.4964	0.4224	0.4206	0.4344	2.3696	0.4213	0.4605	0.4158
0.4238	0.4916	4.8354	0.4210	0.4231	0.4399	0.4155	0.5525	1.5688	0.4128	0.4454	0.5490
0.7202	1.8920	0.4789	2.8487	0.4838	2.2627	0.4311	0.9561	1.4949	0.9930	1.2472	0.4903
0.4695	0.5569	7.8422	0.4527	0.4590	14.5786	0.5591	0.5844	4.2701	0.5491	0.5061	0.5133
0.4418	0.4530	0.4234	0.4579	0.4402	0.4395	0.4360	0.4323	4.3720	0.6030	0.9729	0.9652
0.4440	0.4444	4.3311	0.4792	0.4543	0.4365	0.4367	0.4560	2.4837	0.4485	0.4231	0.4861
1.7348	0.9973	7.5154	5.1946	0.7343	0.4825	0.4365	13.7924	7.1556	0.4541	2.5729	0.4424
0.4454	0.4565	6.1665	0.4642	0.4592	5.8497	0.4374	0.4256	1.2014	0.4394	0.7100	0.4458
0.4343	3.5827	0.4303	1.4966	0.4423	1.0022	0.9625	0.4609	0.4350	0.4470	0.4384	0.4513
0.4295	0.4789	1.2653	0.4466	0.4601	0.4188	0.4233	0.4363	0.4339	0.4424	0.4693	0.4445
0.4214	0.4406	4.0249	0.4737	0.4546	1.7782	0.4658	0.4335	0.4316	0.4500	0.4386	0.4398
0.4661	0.4393	0.4432	0.4456	0.4386	0.4373	0.4407	0.4283	0.4911	0.5003	0.4482	0.4452
0.5060	0.4535	1.2834	0.4328	0.4442	0.4705	0.4601	0.4307	0.4561	0.4977	0.4508	0.4342
0.4388	0.4480	0.4726	0.9694	0.4381	0.5322	0.4431	0.4520	0.4490	0.4303	0.6588	0.4723

y	z
0.4289	0.4092
0.7634	0.4098
0.4243	0.4239
0.5131	0.5114
0.7621	0.5135
0.5419	0.5274
0.4555	0.4648
0.4295	0.4278
0.4432	0.4347
0.4277	0.4425
0.4322	0.4289
0.4456	0.4730
0.6935	0.4165
0.4193	0.4376
0.4496	0.4107
0.5318	0.5107
0.4731	0.4447
0.4730	0.4141
1.2349	0.4389
0.6721	0.4543
0.4531	0.4233
0.4315	0.4535
0.4367	0.4381
0.4455	0.4880
0.4839	0.4468
0.4307	0.4374

References

[1] KH_PrivateBits_CSCW_demo_final, Kirstie Hawkey and Kori M. Inkpen, "PrivateBits: Managing Visual Privacy in the Web Browser", CSCW '06, November 4-8, 2006, Banff, Alberta, Canada.

[2] Michiel Hildebrand, Jacco van Ossenbruggen and Alia Amin, "The Design Space of a Configurable Autocompletion Component", April 21–25, 2008, Beijing, China.

[3] https://en.wikipedia.org/wiki/Autocomplete.

[4] Type Less, Find More: Fast Autocompletion Search with a Succinct Index

[5] Shokouhi-Personalized QAC Milad Shokouhi, "Learning to Personalize Query Auto-Completion", SIGIR'13, ACM 978-1-4503-2034-4/13/07, Dublin, Ireland, July 28–August 1, 2013.

[6] Kyle I. Murray and Jeffrey P. Bigham, "Beyond Autocomplete: Automatic Function Definition", IEEE Symposium on Visual Languages and Human-Centric Computing: Posters and Demos, 2011.

[7] Ming-Jui Huang, Tzao-Lin Lee, "An integrated software processor with autofilling out web forms", 13th Computer Systems Architecture Conference, ACSAC. Asia-Pacific, Page(s) 1 – 8, 4-6 Aug. 2008.

[8] Greg Little and Robert C. Miller, "Keyword programming in Java", March 2009, Volume 16, Issue 1, pp 37-71

[9] Thomas Goldschmidt, Steffen Becker, and Axel Uhl, "Classification of Concrete Textual Syntax Mapping Approaches",Springer-Verlag Berlin Heidelberg 2008.

[10] Neeraj Agrawal Mrutyunjaya Swain, "Auto Complete Using Graph Mining: A Different Approach", Southeastcon, Proceedings of IEEE, ISSN: 1091-0050, Page(s): 268 – 271, Date of Conference 17-20 March 2011.

[11] Oxford English Dictionary Online.

الاكمال التلقائي للكلمات باستخدام المصفوفات المتداخلة

م. م. ايلاف صباح عباس

المستخلص: انظمه التنبؤ بالكلمات تم استخدامها منذ اوائل الثمانيات. وقد تم استخدامها بشكل اصلي من قبل الاشخاص المعاقين لتقليل مقدار الجهد اللازم لادخال النص, ولكن لاحقا وجد انها مفيده ايضا للاشخاص الراغبين بالتعلم او ذوي المشاكل باللغه, مثل الاشخاص اللذين يواجهون مشاكل في التكلم و التهجئ اوقواعد اللغه. الهدف من هذا البحث هو اقتراح نظام اكمال تلقائي للكلمات يعتمد على المصفوفات متعددة الابعاد المتداخله لبناء قاعده بينات تستخدم للخزن المسبق للكلمات ويتم استخدام قاعده البيانات في البحث عن اي كلمه تبدا بحرفين تم ادخالها, حيث انه استغرق النظام وقت مقداره 5.343 ثانيه للبحث عن كلمات تبدأ باول حرفين موجودين في قاعده البيانات و استغرق وت مقداره 2.2568 ثانيه للبحث عن كلمات تبدأ باول حرفين غير موجودين في قاعده البيانات.

الكلمات المفتاحية: متصفح المواقع، الاستعلام الإكمال التلقائي، الاقتراح التلقائي، تصميم الواجهة.

* كلية المنصور الجامعة، قسم هندسة الاتصالات، بغداد، العراق