Using the skeleton app provided in the ***/Interesting Stuff/Hash Tables – Chaining/***folder, complete the implementation of a *Separate chaining with linked lists* collision resolution hash table.

Use the provided input files (100.data, 1000.data etc.) to feed your hash table.

Requirements:

* Implement all the operations as described in the skeleton
* At the end, after every element has been inserted, compute the standard deviation of all the buckets’ sizes. (see [here](http://en.wikipedia.org/wiki/Standard_deviation#Basic_examples) an example of how to compute this) – also, read up on what standard deviation is and try to figure out what a low/high value of such a measure would mean in our case.
* Try things out with a bad hash function first (a H0 – think about the worst hash function which could still somehow distribute entries in more than 1 bucket)

|  |  |  |
| --- | --- | --- |
| **Hash Function (char \* c, int i);** | **ISF \*** | **σ (standard deviation)** |
| 100.data | | |
| H0 | .10 | 9.2736 |
| H1 | .10 | 9.6954 |
| H2 | .10 | 9.4868 |
| H3 | .10 | 9.5917 |
| H0 | .20 | 4.3589 |
| H1 | .20 | 4.7958 |
| H2 | .20 | 4.4721 |
| H3 | .20 | 4.5826 |
| H0 | .35 | 2.4495 |
| H1 | .35 | 2.4495 |
| H2 | .35 | 2.4495 |
| H3 | .35 | 2.4495 |
| H0 | .50 | 1.4142 |
| H1 | .50 | 1.7321 |
| H2 | .50 | 1.7321 |
| H3 | .50 | 1.7321 |
| H0 | .75 | 1.0000 |
| H1 | .75 | 1.0000 |
| H2 | .75 | 1.0000 |
| H3 | .75 | 1.0000 |
| H0 | .85 | 1.0000 |
| H1 | .85 | 1.0000 |
| H2 | .85 | 1.0000 |
| H3 | .85 | 1.0000 |
| 500.data | | |
| H0 | .10 | 9.5917 |
| H1 | .10 | 9.5917 |
| H2 | .10 | 9.9499 |
| H3 | .10 | 9.4868 |
| H0 | .20 | 4.7958 |
| H1 | .20 | 4.5826 |
| H2 | .20 | 5.0000 |
| H3 | .20 | 4.4721 |
| H0 | .35 | 3.6056 |
| H1 | .35 | 2.4495 |
| H2 | .35 | 3.0000 |
| H3 | .35 | 2.4495 |
| H0 | .50 | 3.0000 |
| H1 | .50 | 1.7321 |
| H2 | .50 | 2.6458 |
| H3 | .50 | 1.4142 |
| H0 | .75 | 2.4495 |
| H1 | .75 | 1.0000 |
| H2 | .75 | 2.2361 |
| H3 | .75 | 1.0000 |
| H0 | .85 | 2.4495 |
| H1 | .85 | 1.0000 |
| H2 | .85 | 2.0000 |
| H3 | .85 | 1.0000 |
| 1000.data | | |
| H0 | .10 | 10.2956 |
| H1 | .10 | 9.4868 |
| H2 | .10 | 10.2956 |
| H3 | .10 | 9.5394 |
| H0 | .20 | 7.2801 |
| H1 | .20 | 4.4721 |
| H2 | .20 | 5.8310 |
| H3 | .20 | 4.4721 |
| H0 | .35 | 5.5678 |
| H1 | .35 | 2.4495 |
| H2 | .35 | 4.3589 |
| H3 | .35 | 2.2361 |
| H0 | .50 | 4.6904 |
| H1 | .50 | 1.4142 |
| H2 | .50 | 3.7417 |
| H3 | .50 | 1.7321 |
| H0 | .75 | 3.8730 |
| H1 | .75 | 1.0000 |
| H2 | .75 | 3.0000 |
| H3 | .75 | 1.0000 |
| H0 | .85 | 3.6056 |
| H1 | .85 | 1.0000 |
| H2 | .85 | 2.8284 |
| H3 | .85 | 1.0000 |
| 5000.data | | |
| H0 | .10 | 23.6854 |
| H1 | .10 | 9.4868 |
| H2 | .10 | 19.5192 |
| H3 | .10 | 9.4868 |
| H0 | .20 | 16.7631 |
| H1 | .20 | 5.2915 |
| H2 | .20 | 13.8203 |
| H3 | .20 | 4.4721 |
| H0 | .35 | 12.6886 |
| H1 | .35 | 4.0000 |
| H2 | .35 | 10.4403 |
| H3 | .35 | 2.4495 |
| H0 | .50 | 10.6301 |
| H1 | .50 | 3.3166 |
| H2 | .50 | 8.7750 |
| H3 | .50 | 1.7321 |
| H0 | .75 | 8.6603 |
| H1 | .75 | 2.8284 |
| H2 | .75 | 7.1414 |
| H3 | .75 | 1.0000 |
| H0 | .85 | 8.1240 |
| H1 | .85 | 2.6458 |
| H2 | .85 | 6.7082 |
| H3 | .85 | 1.0000 |
| 10000.data | | |
| H0 | .10 | 33.5857 |
| H1 | .10 | 11.1355 |
| H2 | .10 | 27.8029 |
| H3 | .10 | 9.5394 |
| H0 | .20 | 23.7487 |
| H1 | .20 | 7.8740 |
| H2 | .20 | 19.6723 |
| H3 | .20 | 4.5826 |
| H0 | .35 | 17.9722 |
| H1 | .35 | 5.9161 |
| H2 | .35 | 14.8661 |
| H3 | .35 | 2.4495 |
| H0 | .50 | 15.0333 |
| H1 | .50 | 5.0000 |
| H2 | .50 | 12.4499 |
| H3 | .50 | 1.7321 |
| H0 | .75 | 12.2882 |
| H1 | .75 | 4.1231 |
| H2 | .75 | 10.1980 |
| H3 | .75 | 1.0000 |
| H0 | .85 | 11.5326 |
| H1 | .85 | 4.0000 |
| H2 | .85 | 9.8489 |
| H3 | .85 | 1.0000 |
| 25000.data | | |
| H0 | .10 | 53.1977 |
| H1 | .10 | 17.8885 |
| H2 | .10 | 44.1022 |
| H3 | .10 | 9.5394 |
| H0 | .20 | 37.6165 |
| H1 | .20 | 12.6491 |
| H2 | .20 | 31.1929 |
| H3 | .20 | 4.5826 |
| H0 | .35 | 28.4429 |
| H1 | .35 | 9.5917 |
| H2 | .35 | 23.5797 |
| H3 | .35 | 2.4495 |
| H0 | .50 | 23.7908 |
| H1 | .50 | 8.0000 |
| H2 | .50 | 19.7231 |
| H3 | .50 | 1.7321 |
| H0 | .75 | 19.4422 |
| H1 | .75 | 6.5574 |
| H2 | .75 | 16.1245 |
| H3 | .75 | 1.0000 |
| H0 | .85 | 18.2483 |
| H1 | .85 | 6.1644 |
| H2 | .85 | 15.1327 |
| H3 | .85 | 1.0000 |
| 50000.data | | |
| H0 | .10 | 75.3060 |
| H1 | .10 | 25.4362 |
| H2 | .10 | 62.4340 |
| H3 | .10 | 9.4868 |
| H0 | .20 | 53.2541 |
| H1 | .20 | 18.0000 |
| H2 | .20 | 44.1475 |
| H3 | .20 | 4.5826 |
| H0 | .35 | 40.2616 |
| H1 | .35 | 13.6015 |
| H2 | .35 | 33.3766 |
| H3 | .35 | 2.4495 |
| H0 | .50 | 33.6898 |
| H1 | .50 | 11.4018 |
| H2 | .50 | 27.9285 |
| H3 | .50 | 1.7321 |
| H0 | .75 | 27.5136 |
| H1 | .75 | 9.3274 |
| H2 | .75 | 22.8035 |
| H3 | .75 | 1.0000 |
| H0 | .85 | 25.8457 |
| H1 | .85 | 8.7750 |
| H2 | .85 | 21.4243 |
| H3 | .85 | 1.0000 |
| 100000.data | | |
| H0 | .10 | 106.5317 |
| H1 | .10 | 36.1248 |
| H2 | .10 | 88.3289 |
| H3 | .10 | 9.5394 |
| H0 | .20 | 75.3326 |
| H1 | .20 | 25.5539 |
| H2 | .20 | 62.4580 |
| H3 | .20 | 4.4721 |
| H0 | .35 | 56.9473 |
| H1 | .35 | 20.8567 |
| H2 | .35 | 51.0000 |
| H3 | .35 | 2.8284 |
| H0 | .50 | 47.6445 |
| H1 | .50 | 16.1555 |
| H2 | .50 | 39.5095 |
| H3 | .50 | 1.7321 |
| H0 | .75 | 38.9102 |
| H1 | .75 | 13.1909 |
| H2 | .75 | 32.2645 |
| H3 | .75 | 1.0000 |
| H0 | .85 | 36.5513 |
| H1 | .85 | 12.4097 |
| H2 | .85 | 30.2985 |
| H3 | .85 | 1.0000 |

\* ISF = Initial Size Factor

Have other combinations in mind? Feel free to fill-up the table with more tries to see if any interesting results come up!

H0: bad hash function!

H1:

***int hashFunction(char \* content, int i)***

***{***

***int length = strlen(content);***

***int k, sum;***

***for (sum=0, k=0; k < length; k++)***

***{***

***sum += content[k];***

***}***

***return sum % size;***

***}***

H2 – H3 🡪 your choices!

* Try to improve each time

Deadlines:

30411 – 18.05.2015 (before 12:00 if you want review and before 23:59 if you want a grade)  
30414 – 19.05.2015 (before 12:00 if you want review and before 23:59 if you want a grade)