CSI_4_DSA

DATA STRUCTURES AND ALGORITHMS COURSE WORK

Doan Gia Khanh Nguyen
STUDENT ID: 4328064
LECTURER: MIKE CHILD

Table of content

Figure list

Task 1

Hashset

hashset N	10000	10544	11117	11722	12360	13033	13742	14490	15278	16109	16986	17910	18885	19912
20996	22138	23343	24613	25952	27365	28853	30423	32079	33824	35665	37606	39652	41809	44084	46483
49012	51679	54491	57456	60582	63879	67355	71020	74884	78959	83255	87785	92561	97598	102908	108508
114412	120637	127202	134123	141421	149116	157230	165785	174805	184317	194346	204921	216071	227828	240224	253295
267078	281610	296933	313090	330126	348088	367029	386999	408057	430260	453671	478356	504385	531829	560767	591279
623452	657375	693144	730860	770627	812558	856771	903390	952545	1004375	1059025	1116649	1177408	1241473	1309023	1380250
1455352	1534540	1618038	1706078	1798909	1896791										
hashset T	4	2	3	2	2	2	2	1	0	1	2	0	0	1
1	1	1	0	1	1	1	0	1	1	1	1	1	1	1
1	1	0	0	1	1	1	0	1	0	0	0	1	0	0	0
0	1	1	1	1	1	1	1	0	1	1	0	0	1	1	1
1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	2	1	2	1	1	1	1	1	2	1	1
1	2	1	1	2	2										

Figure 1. HashSet

hashset N	10000	10544	11117	11722	12360	13033	13742	14490	15278	16109	16986	17910	18885	19912
20996	22138	23343	24613	25952	27365	28853	30423	32079	33824	35665	37606	39652	41809	44084	46483
49012	51679	54491	57456	60582	63879	67355	71020	74884	78959	83255	87785	92561	97598	102908	108508
114412	120637	127202	134123	141421	149116	157230	165785	174805	184317	194346	204921	216071	227828	240224	253295
267078	281610	296933	313090	330126	348088	367029	386999	408057	430260	453671	478356	504385	531829	560767	591279
623452	657375	693144	730860	770627	812558	856771	903390	952545	1004375	1059025	1116649	1177408	1241473	1309023	1380250
1455352	1534540	1618038	1706078	1798909	1896791										
hashset T	42	20	21	22	17	25	21	22	22	23	23	22	22	23
23	23	23	23	24	19	19	19	19	26	21	21	22	22	22	23
23	20	22	20	20	20	21	21	21	21	21	23	22	22	22	22
22	23	23	23	23	23	23	23	22	23	23	30	24	26	25	27
25	25	27	34	36	28	29	32	36	34	39	34	40	37	36	36
37	35	44	45	44	51	49	46	59	72	61	63	67	68	66	81
68	89	75	75	76	82										

Figure 2. HashSet 500000

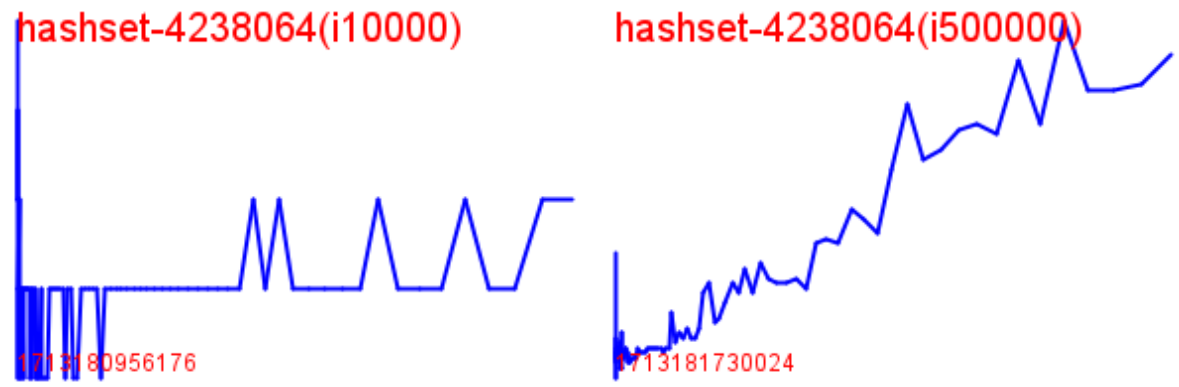


Figure 3. HashSet Graph

Figure 4. HashSet Graph 500000

1. The underlying data structure in this case is Hashtable.
2. Analyzing the big O performance
   a. Adding an item that does not exist in the data structure

   It is generally O(1) because the time operation is constant to insert the element to the set.

   b. Removing an item from the data structure

   It will take O(1) time to located the element need to remove and the collision is expected to be small which make the time complexity constant.

c. Attempting to add an existing item to a set or changing the value of an existing mapping

The expected time complexity is O(1) for both scenarios because the item already exist in hash set so the set will able to find them immediately.

3. Discuss the graph

   The graph fluctuate all the period but the trend suggest the hash set operation remain constant. This reflect closely to the expectation and match the theoretical performance.

4. Compare the graph

   There is a huge different in two figure, figure 4 shows a dramatically grow from the start to the end but there are some similar pattern in the fluctuation. The time complexity for both of them are same to each other at O(1).

Listset

listset N	100	105	111	117	123	130	137	144	152	161	169	179	188	199
209	221	233	246	259	273	288	304	320	338	356	376	396	418	440	464
490	516	544	574	605	638	673	710	748	789	832	877	925	975	1029	1085
1144	1206	1272	1341	1414	1491	1572	1657	1748	1843	1943	2049	2160	2278	2402	2532
2670	2816	2969	3130	3301	3480	3670	3869	4080	4302	4536	4783	5043	5318	5607	5912
6234	6573	6931	7308	7706	8125	8567	9033	9525	10043	10590	11166	11774	12414	13090	13802
14553	15345	16180	17060	17989	18967									
listset T	0	1	0	1	0	0	1	0	0	1	0	1	0	1
0	1	0	0	1	1	1	0	0	1	1	0	1	0	1	0
0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	1
0	1	1	0	0	1	1	0	0	1	1	0	1	1	0	0
1	0	1	0	0	0	0	0	0	1	0	1	1	1	1	0
1	1	1	1	1	2	2	2	2	1	1	2	2	1	2	2
2	3	4	2	2	2									

Figure 5. ListSet

listset N	100	105	111	117	123	130	137	144	152	161	169	179	188	199
209	221	233	246	259	273	288	304	320	338	356	376	396	418	440	464
490	516	544	574	605	638	673	710	748	789	832	877	925	975	1029	1085
1144	1206	1272	1341	1414	1491	1572	1657	1748	1843	1943	2049	2160	2278	2402	2532
2670	2816	2969	3130	3301	3480	3670	3869	4080	4302	4536	4783	5043	5318	5607	5912
6234	6573	6931	7308	7706	8125	8567	9033	9525	10043	10590	11166	11774	12414	13090	13802
14553	15345	16180	17060	17989	18967									
listset T	8	6	3	4	3	2	2	1	2	2	2	1	2	2
2	2	2	2	2	2	2	3	3	2	2	3	3	4	4	4
5	6	4	5	6	5	5	6	5	6	6	7	7	8	7	9
9	15	13	11	11	12	13	14	14	15	15	16	17	18	17	23
26	26	30	25	29	27	30	34	34	34	40	41	39	39	41	44
47	49	53	54	71	73	69	73	77	87	83	92	95	104	104	124
122	126	121	124	137	140									

Figure 6. ListSet 500000

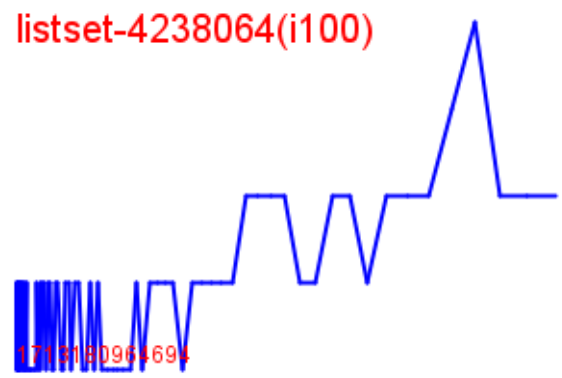


Figure 7. ListSet Graph

Figure 8. ListSet Graph 500000

1. The underlying data structure in this case is a list-based data structure.
2. Analyzing the big O performance
   a. Adding an item that does not exist in the data structure

   The complexity is O(n) because element need to be processed to show its uniqueness.

   b. Removing an item from the data structure

   Same with addition, it take O(n) time.

   c. Attempting to add an existing item to a set or changing the value of an existing mapping

   The time complexity is O(n).

3. Discuss graph

   Figure 7 shows a stable in almost all the period and rise at the end of it. The expectated time complexity for this figure is O(1) because the time taken does not significantly increase with the ionput size.

4. Compare the graph

   In the figure 8, it is clearly seen that this graph is a linear. And the time complexity for it will be expect at O(n) which will be lower than the O(1) time of figure 7.

Treeset

treeset N	10000	10544	11117	11722	12360	13033	13742	14490	15278	16109	16986	17910	18885	19912
20996	22138	23343	24613	25952	27365	28853	30423	32079	33824	35665	37606	39652	41809	44084	46483
49012	51679	54491	57456	60582	63879	67355	71020	74884	78959	83255	87785	92561	97598	102908	108508
114412	120637	127202	134123	141421	149116	157230	165785	174805	184317	194346	204921	216071	227828	240224	253295
267078	281610	296933	313090	330126	348088	367029	386999	408057	430260	453671	478356	504385	531829	560767	591279
623452	657375	693144	730860	770627	812558	856771	903390	952545	1004375	1059025	1116649	1177408	1241473	1309023	1380250
1455352	1534540	1618038	1706078	1798909	1896791									
treeset T	4	6	4	4	3	3	3	3	3	3	3	3	3	3
3	3	4	4	3	4	3	4	3	4	4	3	3	2	3	2
3	3	2	3	3	3	4	4	4	4	3	3	4	2	3	2
4	3	3	3	3	2	3	5	3	5	4	4	3	5	3	4
4	3	4	4	3	4	6	4	5	4	5	4	5	5	5	5
5	5	4	4	5	6	6	7	6	5	6	6	5	7	6	6
7	7	6	7	6	7									

Figure 9. TreeSet

treeset N	10000	10544	11117	11722	12360	13033	13742	14490	15278	16109	16986	17910	18885	19912
20996	22138	23343	24613	25952	27365	28853	30423	32079	33824	35665	37606	39652	41809	44084	46483
49012	51679	54491	57456	60582	63879	67355	71020	74884	78959	83255	87785	92561	97598	102908	108508
114412	120637	127202	134123	141421	149116	157230	165785	174805	184317	194346	204921	216071	227828	240224	253295
267078	281610	296933	313090	330126	348088	367029	386999	408057	430260	453671	478356	504385	531829	560767	591279
623452	657375	693144	730860	770627	812558	856771	903390	952545	1004375	1059025	1116649	1177408	1241473	1309023	1380250
1455352	1534540	1618038	1706078	1798909	1896791									
treeset T	157	89	84	82	80	86	81	72	76	74	75	101	76	78
79	80	77	85	83	88	85	83	96	89	93	92	106	96	92	92
100	105	96	103	102	104	106	99	106	111	103	94	114	106	110	112
110	114	118	113	119	117	120	117	119	127	121	129	128	127	127	129
130	131	134	134	135	136	141	161	140	166	203	165	154	157	163	172
181	178	183	184	186	168	200	202	204	211	214	222	235	232	237	253
242	253	254	242	294	292									

Figure 10. TreeSet 500000

Figure 11. TreeSet Graph

Figure 12. TreeSet Graph 500000

1. The underlying data structure in this case is a tree set.
2. Let's analyze the big O performance for each operation
   a. Adding an item that does not exist in the data structure

   The time complexity for this case is O(log n) because it use a binary search tree structure.

   b. Removing an item from the data structure

   It also take O(log n) time to execute this action

   c. Attempting to add an existing item to a set or changing the value of an existing mapping

The tree set will know the item already exists in the set so the time complexity still O(log n).

3. Discuss graph

   The figure 11 undulated in whole period and reached it peak at the end. The time taken is increase gently as the size of the tree set increase.

4. Compare the graph

   The figure 12 look different compare to the previous figure where the growth rate levelled off, time taken and the size of tree set increase slowly. And the time complexity of figure 12 will smaller than figure 11.

Task 2

Program

```
import dsa.cw2324.WikiFetcher;
import java.io.IOException;
import java.util.*;
class wikiCoursework {
    public static void main(String[] args) throws IOException   {
        TreeMap<String, Map<String, Integer>> outermap = new TreeMap<>();
        for (int i=0; i<10; ++i){
            WikiFetcher wf = new WikiFetcher();
            String[] words = wf.getWords();
            String url = wf.getUrl();
            TreeMap<String, Integer> frequencymap = new TreeMap<>();

            for (String word: words){
                if (!frequencymap.containsKey(word)){
                    frequencymap.put(word, 1);
                }else {
                    Integer new_num = frequencymap.get(word) + 1;
                    frequencymap.put(word, new_num);
                }
                TreeMap<String, Integer> thefrequencymap = new TreeMap<>();
                for (String j : frequencymap.keySet()){
                    if (frequencymap.get(j)>9){
                        thefrequencymap.put(j, frequencymap.get(j));
                    }
                }
                outermap.put(url, thefrequencymap);
            }
        }
        WikiFetcher.deepPrint(outermap);
    }
}
```

Figure 13. Task 2 code

Line 1-3: import all the necessary library.

Line 4: creating the class wikiCoursework .

Line 5: the main function.

Line 6: create the outer map, key is url and value is the frequency.

Line 7: generate a 10 times for loop.

Line 8-10: get the word and the url by using WikiFetcher.

Line 11: create a new frequency map, new key is the word and new value is the frequency of that word.

Line 13: for loop iterate over each word array.

Line 14-15: check if the frequency map does not have the current word and add it to the frequency map with the frequency of 1.

Line 16-18: else statement to increase the word frequency from the frequency map by 1, assign value to new_num and update the current frequency.

Line 20: declares a new tree map named the frequency map.

Line 21: : for loop iterate over each word in the frequency map.

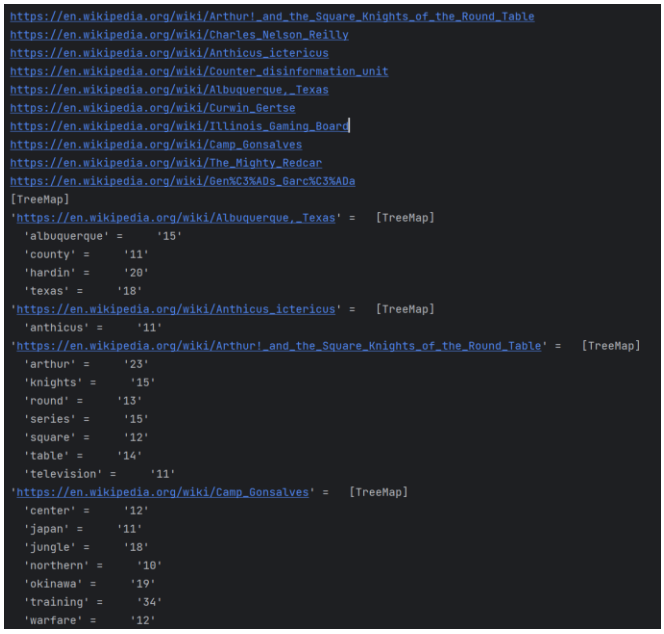
Line 22: check if that word frequency is greater than 9.

Line 23: add the word and it frequency to the frequency map.

Line 26: add the frequency map to the outer map, the key is url.

Line 29: print out the result.

Output



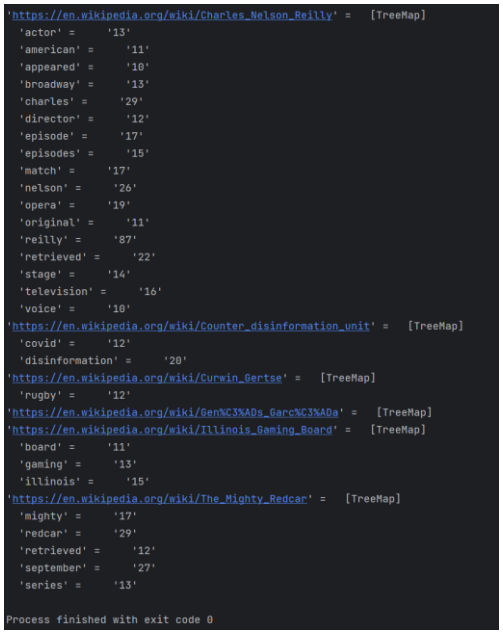


Figure 14. Task 2 result #1

Figure 15. Task 2 result #2

Task 3

Program

```
x = set()
y = set()

for i in range(100):
    n = nextNumber()
    if n in x:
        y.add(n)
    else:
        x.add(n)
u = len(x)
d = len(y)
print("list of ",u,"unique number:")
print(x)
print("list of ",d,"duplicate number:")
print(y)
```

Figure 16. Task 3 code

Firstly, understanding the nextNumber() function will be important because it will give the program in the brief of a thousand numbers between 0 and 1000 and ask the user to enter their student number. Secondly, I generate x as a group of unique number and y as a group which contain duplicate number. Next, I use the for loop to create 100 random numbers through the nextNumber() function. In order to separate them in group x and group y, the code check the existence of the number in unique group, if it is already there, that number will be add to the group for number that occurred more than once. In the other hand, if it is not in x group, that number will be count as a unique number and add in x group. Thirdly, u and d are the length of unique number list and the length of duplicate number list, respectively. Finally, use print to bring the result of the for each group out the screen.

Output

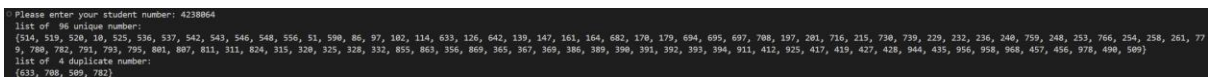


Figure 17. Task 3 result

References

Phyex (2022) What is the complexity of cointains method of HashSet? [reddit], 7 December. Access avaiable at: https://www.reddit.com/r/java/comments/rb24ks/what_is_the_complexity_of_contains_method_of/

Eran (2017) Java why does HashSet's remove() take O(1) time, when ArrayList's remove() take O(n)? [staskoverflow], 3 December. Access avaiable at: https://stackoverflow.com/questions/47620443/java-why-does-hashsets-remove-take-o1-time-when-arraylists-remove-takes

Baeldung (18 March 2024) Guide to Scala ListSet. Avaiable at: https://www.baeldung.com/scala/listset

Oleksandr Miadelets (18 May 2023) TreeSet in Java. Avaiable at: https://codegym.cc/groups/posts/treeset-in-java