Experimentation

Student ID:961597

Student Name: Guang Yang

Student account: GYYA@student.unimelb.edu.au

1. Accuracy test

This test will include two parts. The first is the blank test, and the second is the small size normal test which will be checked manually.

1). blank test

① If the input file name is blank/ if the input file is not exist

For dict1, the output is “failed to open file for reading”, and nothing is wrote in the output file.

For dict2, the output is “failed to open file for reading”, and nothing is wrote in the output file.

The same as my expectation.

② If the input file is blank

For dict1, the output is “dictionary is not available!”, and nothing is wrote in the output file.

For dict2, the output is “dictionary is not available!”, and nothing is wrote in the output file.

The same as my expectation.

③ If the output file name is blank

For dict1, the output is “failed to open file for writing”.

For dict2, the output is “failed to open file for writing”.

The same as my expectation.

④ If the search key file is blank

For dict1, the output is nothing.

For dict2, the output is nothing.

The same as my expectation.

2) . small size normal test

I get 10 lines form the test file. Their names are listed below.

Sofiya Aleksandrovna Velikaya

Toshiaki Kitai

Iera Echebarra Fernndez

Adalbert Dickhut

James Reid

Reza Mohammad Ali Yazdani

Ernst Theodor Georgsson Westerlund

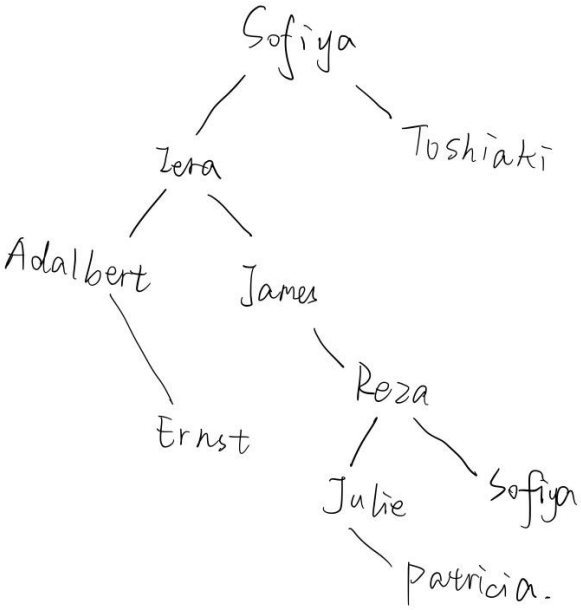
Julie Pomagalski

Patricia Chauvet-Blanc

Sofiya Aleksandrovna Velikaya

This is the tree I draw manually which is based on the algorithm of dict1.

Aa	Bb	Cc	Dd	Ee	Ff	Gg
Hh	Ii	Jj	Kk	Ll	Mm	Nn
Oo	Pp	Qq		Rr	Ss	Tt
Uu	Vv	Ww		Xx	Yy	Zz

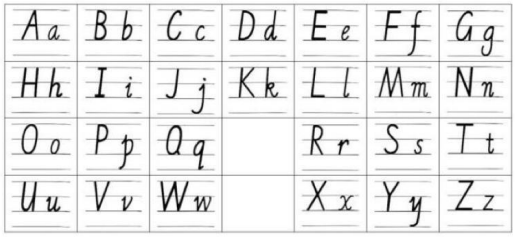


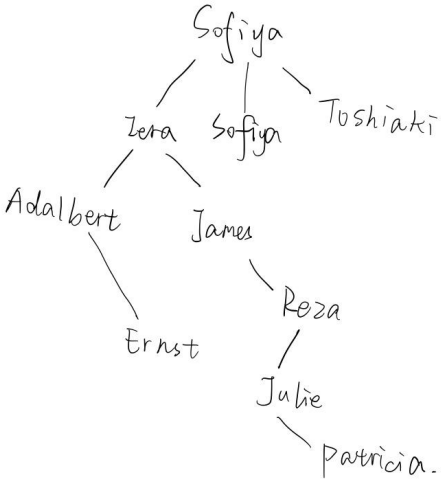
And the result of my program is below.

```
-bash-4.1$ dict1 test2.csv 1.txt< key2.txt
Sofiya Aleksandrovna Velikaya --> 5
Toshiaki Kitai --> 2
Iera Echebarra Fernndez --> 4
Adalbert Dickhut --> 3
James Reid --> 3
Reza Mohammad Ali Yazdani --> 6
Ernst Theodor Georgsson Westerlund --> 4
Julie Pomagalski --> 5
Patricia Chauvet-Blanc --> 6
```

The stdout is right. And the output file is too long to paste in this report but it's the same as my expectation.

For stage2, the expectation and result is shown below.






```
-bash-4.1$ dict2 test2.csv 1.txt< key2.txt
Sofiya Aleksandrovna Velikaya --> 1
Toshiaki Kitai --> 2
Iera Echebarra Fernndez --> 2
Adalbert Dickhut --> 3
James Reid --> 3
Reza Mohammad Ali Yazdani --> 4
Ernst Theodor Georgsson Westerlund --> 4
Julie Pomagalski --> 5
Patricia Chauvet-Blanc --> 6
```

The result is also right. Thus, this program is accurate.

2. Complexity analyse

In this part, I'll find 500 names randomly as the key to search in different dictionaries and to verify the suppose of big O theory. Dict2 algorithm will be equal to or better than Dict1.

For the binary search tree, good case behavior is logn, worst case behavior is n. This tree is not perfectly balanced, Thus I think the result of stage1 and stage2 is between O(n) and O(lgn).

If the dictionary(contains about 270000 lines) is inserted randomly, the average compare times number is 26.862000 based on the algorithm of dict1.

```
average is 26.862000
```

If the dictionary(contains about 270000 lines) is inserted in sorted order (Ascending), the average compare times number is 38088.859375 based on the algorithm of dict1.

```
average is 38088.859375
```

If the dictionary(contains about 270000 lines) is inserted in sorted order (Descending), the average compare times number is 68765.539062 based on the algorithm of dict1.

```
average is 68765.539062
```

If the dictionary(contains about 270000 lines) is inserted randomly, the average compare times number is 21.563999 based on the algorithm of dict2.

```
average is 21.563999
```

If the dictionary(contains about 270000 lines) is inserted in sorted order (Ascending), the average compare times number is 38082.867188 based on the algorithm of dict2.

```
average is 38082.867188
```

If the dictionary(contains about 270000 lines) is inserted in sorted order (Descending), the average compare times number is 34440.511719 based on the algorithm of dict2.

```
average is 34440.511719
```

If the dictionary(contains about 130000 unique names) is inserted in sorted order(Ascending), the average compare times number is 38083.050781 based on the algorithm of dict1.

```
average is 38083.050781
```

If the dictionary(contains about 130000 unique names) is inserted in sorted order(Descending), the average compare times number is 34441.507812 based on the algorithm of dict1.

```
average is 34441.507812
```

If the dictionary(contains about 130000 unique names) is inserted in sorted order(Ascending), the average compare times number is 38082.859375 based on the algorithm of dict2.

```
average is 38082.859375
```

If the dictionary(contains about 130000 unique names) is inserted in sorted order(Descending), the average compare times number is 34440.511719 based on the algorithm of dict2.

```
average is 34440.511719
```

In order to make this data more readable, I make a table to show my result.

	random	sorted(ascending)	sorted(descending)
stage1	26.862	38088.85938	68765.53906
stage2	21.563999	38082.86719	34440.51172

	unique and sorted(ascending)	unique and sorted(descending)
stage1	38083.05078	34441.50781
stage2	38083.05078	34441.50781

Conclusion: According to the result, the inserted randomly dictionaries are tending to have better representation than the inserted in sorted order dictionaries.

Because $log_2 270000 \approx 18$, I think the complexity of inserted randomly dictionaries(26 and 21) is almost O(lgn). But the inserted in sorted order dictionaries are tending to have O(n) complexity no matter it is sorted ascending or descending.

In the algorithm of dict1, ascending insert and descending insert have obvious difference(38088 ascending and 68765 descending). This is mainly because we just search for the left child for same key. But this change has little influence in finding keys in dict2(38082and34440), because the algorithm of dict2 have a new memory to store the same key, it doesn't need to search to either left or right side if they have found the key.

When we delete the duplicating key in the dictionary, we find out that there are basically no difference between algorithm1 and 2 in no matter ascending or descending order.

Therefore, my suppose is confirmed. For the randomly inserted almost balanced tree, the complexity of search in this dictionary is O(logn) no matter it's in stage1 or stage2 algorithm. But in the worst case tree, the complexity is O(n). Tough 30000 may less than 270000, 2^30000 is far beyond 270000. The average case of not balanced binary search tree will be O(n). If the key is unique, there are no difference between algorithm1 and algorithm2, but if there are some duplicates, algorithm2 will work better than algorithm1, but algorithm2 uses more memory than algorithm2.