**Name: Weicheng Ao**

This file has been provided to help you structure your thoughts when discussing Lab 3, Part c with your marker. Some questions can be answered with a single sentence, some may require much longer answers. You are free to edit/rearrange this file as much as you want.

All questions should be answered for **each** of your chosen data structures.

------------------------

**Initial Expectations**

------------------------

**>> Do you expect this data structure to be preferable to the others on all inputs, most inputs, some inputs? Why?**

1. Dynamic Array with Sorting:

This data structure is more preferable to the others on **some inputs**, and when the inputs' pattern more diverse, seems more random. And very good **for the inputs number that is not exceeds the array size**. (This is because the Dynamic Array is a fixed size initially, if there are no space available, we need to expand the size by a factor of two normally, **this costs a lot of memory in the scale of O(n). But for the access it takes O(1) time complexity on average to find a particular element**.)

2. Binary Trees:

This data structure is more preferable to the others **on most inputs**, as it constructs by **using orders**.

3. Hash Set with Linear Probing:

This data structure is more preferable to the others on **some inputs**, and **for these inputs are not repeated for a hash value.**

4. Hash Set with Separate Chaining:

This data structure is more preferable to the others **on most inputs** as in this case, the **rehashing could become less often**.

But, **Hash Set need to rehash** when the load factor reaches a specific level, this means **I assume for the large amount of data, Binary Search Tree is preferable to the others**.

**>> Do you expect your answer to change if the order of the words in your input dictionary is in the best/worst case? Why?**

In the **best case**, we expect every implementation takes **O(1) time**, this is a constant, hence **in this case, we can use any structure.**

In the **worst case**, we need to **consider these** **implementations one by one**.

**For binary trees**, the **worst case** happens when the inputs are monotonus, **say gradually larger or gradually smaller.** In such case, the tree is build from one side, say right or left. So either inserting or searching **takes O(n) time.**

For **dynamic array with sorting performed**, the worst case arises **when the sorting needs to take most of the time**, when the data is descending ordered, say **O(n^2) for insertion sort, the comparison could be 0.5n\*(n-1) times.**

For **hash set with linear probing**, the worst case is where **all data evaluate to the same hash value**, this makes the insertion and finding more memory demanding and degrades to the linear search where the complexity is **O(n).**

For **hash set with separate chaining**, the worst case is also when **all data evaluate to the same hash value**, it then becomes linear search, which takes **O(n)** time.

All in all**, in terms of performance in the worst cases as big O notation presented**, I prefer to use **binary search trees, hash set with linear probing and hash set with separate chaining** instead of using dynamic array with sorting (when the sorting behaves badly).

**>> Can you phrase what you expect in terms of a one or two sentence hypothesis that you can test?**

In terms of **running times**:

For the **worst case**, I expect binary search trees, hash set with linear probing and hash set with separate chaining **performs nearly the same** and their running times would be **approximately the same**, the **dynamic array would be cost a little more amount of time**. For the **best case**, I expect **hash set performs better than the others**. For **all random generated inputs**, we need to **consider their average complexity cases**, but I expect the binary search tree performs a little faster.

------------------------

**Experimental Design**

------------------------

**>> How are you going to define what it means for one data stucture to be preferable to another?**

In terms of time. If one data structure is preferable to another, it will be reflected through its timing. The better it behaves, the faster it runs and spend less time to finish.

**>> Which conditions will you vary in your experiment?**

1. **Different size of dictionaries** (simple or large) **or input patterns** (more or less replicates, ordered or not).
2. **Different size of hash tables** to begin with for Hash Set Implementations.
3. **Different size of Finds (Queries) and Different percentage of true values.**

**>> How will you vary them? Why did you make these choices? Did you use theoretical complexities, best, worst and average cases to inform your decisions?**

I am going to generate different amount of inputs for each of the data structures, for each input pattern, the inputs are ordered differently dedicately designed for testing their best cases ,worst cases and their average cases. Indeed I use their theoretical complexities to generate different input pattern, this may be able to reflect their truly performances.

**>> How will you generate the data for your experiments?**

**Through a data generator, or manual type data in a file.**

I will **vary the amount of inputs first for each different data structures and then vary the pattern of inputs (asending or desending or random)** for each of them.

**>> How will you validate your findings?**

I will **repeat the experiments seveal times for each test**, in order to find an average approximated value.

------------------------

**Results and Analysis**

------------------------

**>> What results did you record?**

Experiment 1: Running Times Comparison by running different amount of data.

Small Amount of Data 1. (Keep other variables unchanged and use the same data set)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Data Structures | Amount of Data Used | String Length | Initial Length of Hash Set | Running Times in Sec | | | No. of Collisions for Hash Set | The average number of comparisons per insertion or find for BST |
| Binary Search Tree | 100 | 40 | 10 | 0.08 | 0.07 | 0.07 | - | 7.61 |
| Dynamic array with sorting performed | 0.08 | 0.08 | 0.08 | - | - |
| hash set with separate chaining | 0.12 | 0.66 | 0.66 | 1137 | - |

Small Amount of Data 2. (Keep other variables unchanged and use the same data set)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Data Structures | Amount of Data Used | String Length | Initial Length of Hash Set | Running Times in Sec | | | No. of Collisions for Hash Set | The average number of comparisons per insertion or find for BST |
| Binary Search Tree | 1000 | 40 | 10 | 0.15 | 0.15 | 0.16 | - | 12.172 |
| Dynamic array with sorting performed | 0.10 | 0.11 | 0.10 | - | - |
| hash set with separate chaining | 0.13 | 0.14 | 0.14 | 2134 | - |

Large Amount of Data. (Keep other variables unchanged and use the same data set)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Data Structures | Amount of Data Used | String Length | Initial Length of Hash Set | Running Times in Sec | | | No. of Collisions for Hash Set | The average number of comparisons per insertion or find for BST |
| Binary Search Tree | 10000 | 49 | 10 | 0.96 | 1.01 | 0.99 | - | 15.9678 |
| Dynamic array with sorting performed | 1.32 | 1.32 | 1.35 | - | - |
| hash set with separate chaining | 0.81 | 0.81 | 0.80 | 16350 | - |

**>> What does this tell you about the performance of the data structure?**

**When the amount of data is small, we can see that the Binary Search Tree has the best running times.**

**As the data size gradually to increase, we can see that Dynamic array with some sorting performed takes this advantage while the Binary Search Tree now becomes the worst one.**

**Increasing the data size further, we can see that hash set with separate chaining performs the best of it, and dynamic array with sorting involved droped its position to the worst one.**

**>> What is the answer to the question "Under what conditions is it preferable to use this data structure?"**

**When the size of data is small, say < 500, we can see that the Binary Search Tree has the best running times, thus it is preferable in this case.**

**When the size of data is medium, say < 5000, we can see that the Dynamic array with some sorting involved has the best running times, thus it is preferable in this case.**

**When the size of data is large, say >= 10000, we can see that the hash set with separate chaining has the best running times, thus it is preferable in this case.**