COSC 1252 C++ Programming

Assignment 1

Data Structure Report

S3560808 Kai Zhang

1. Introduction of this report

In modern programming technologies, there's not doubts that at most time programmer uses Standard Template Library (abbr. STL) as a container instead write own data structures. Though, with development of computer science, computers or even microcomputers are powerful enough to run a problem with different data structure and use the same time. However, it is just user's feeling or simple problem. When we encounter problem with large input size or complicated problem. Use which data structure or whether needs to write own data structure needs careful consideration.

1.1 Data structures will be analyzed in this report

1) List (From STL)
2) Vector (From STL)
3) Multiset (From STL)
4) Single list (Self implemented)
5) Binary search tree (Self implemented)

1.2 Design of evaluation

No matter which data structure we are going to analysis. We need an application which implements five different data structures. In this case we use the application which wrote in assignment 1, "String finding and comparing".

Moreover, I will input same volume of dictionary words but different volumes of passage length (12000, 24000 and 48000) to see how data structure will influence application implementation.

Furthermore, timing will be mainly splitting into two different parts: insert time and get & implementation time.

2. Data structure analysis from theory

1) List (From STL)

List, which is double linked, is the simplest data structure from STL. List have several characteristics:

- Lower insert cost, no matter inserts in middle or at the end
- Slower retrieve time
- Does not have real size
- Memory not continuously allocated

This mainly because list does not have contiguous memory. When we insert a node into a list, system will randomly give it a location in memory to store this node and link previous node and last node. Furthermore, we can find that it is cheap to insert. However, for the memories of list are not continuously allocated. So, link retrieve next element by node pointer, which consumes times.

- Need more space

Moreover, as mentioned above. It uses pointer to link! Thus, it actually need a little more space than vector to store information.

- Only get from first element

More important, list only can go forward and backward, so it cannot be accessed the elements from middle only can iterate from first!

- Not synchronized

May cause risk of data corruption when multiple threads reach the list

Summary:

List is suitable for applications those need to add elements frequently cause it both time costless and space costless. Like merge sort, need always add and combine elements several times. Or insert sort need make a new sequence and compare them from first element then need insert in half of sequence frequently, which operation are much cheaper in list than in vector.

For this string compare application, we estimated that it will be much quicker than vector when we add load words in to memory however there will be a significant slower when we retrieve words and word compare stage.

2) Vector (From STL)

Vector is much more like an array, which is quite like list from STL. They at both a sequence of space to store information. But what the difference between them!? Explore principle behind is quite important.

For junior programmers, vector can access information or node like an array using [], which means it can execute much more algorithm which need to start from middle of the sequence to contribute to time efficiency. Like binary search and quick sort.

However, in most application we do not usually retrieve data from middle. So, the features what vector has is most important part that we need consider when we choose a data structure.

- Insert in middle is costly for vector
- Has quicker access speed than list
- All data's memory locations are contiguous

Contiguous memory is the core idea of Vector, because of this special trait vector can iterate vector with extremely high speed for vector does not use pointer. However, coin has two sides, when we want to insert some info into the middle of vector, speed will dramatically go down. Vector has to move all element after insert point backward, which deduce time efficiency.

- Need to get volume of words previously
- Or more space

Though unlike list needs more space not for information itself but for pointer, vector may also require more space than information. For normally size of input cannot be predicted precisely and vector needs contiguous memory. Once input exceeded initial volume, whole vector needs to be relocated which really consumes time efficiency and space efficiency.

Last but not the least, it is synchronized, which means we can process it with multithreads.

Summary:

Vector are suitable for algorithm those need random access to a sequence of data, or for the data are already sorted and do not always insert info in the middle.

For this application, vector will insert far slower than list. On other side, compare progress will be much quicker

3) Multiset (From STL)

When we come to talk about Multiset things becomes far more interesting. The reason is that it is not only just related to how it is stored in memory, but also how the structure is implemented! Every time, when an element is inserted into a multimap it is sorted then put into a special place and the structure behind is a red black tree, which is kind of binary search tree. Fairly quicker for find a string than list, vector.

Summary:

There is no doubt that this structure is good for data that is increasing growing. In this string compare application, insertion should be takes time however

4) Single list (Self implemented)

Single list is an easy version of list, which can only retrieve next node, and also take less cost while user insert into middle of whole list. Generally, it will have same speed with STL list while insert and get.

Summary:

Theoretically, it means it will be slower while comparing than vector and multiset but have faster insert speed than over data structure.

5) Binary search tree (Self implemented)

Though it quite like principle behind multiset, it will be slower than multiset. This is mainly because BST does not

restructure the tree, binary search tree not always a balanced tree. Furthermore, this need self-defined comparison algorithm. Different algorithm takes different times to structure the tree.
Summary
In my implementation, I used Levenshtein compare algorithm which will be much slower than common algorithm.

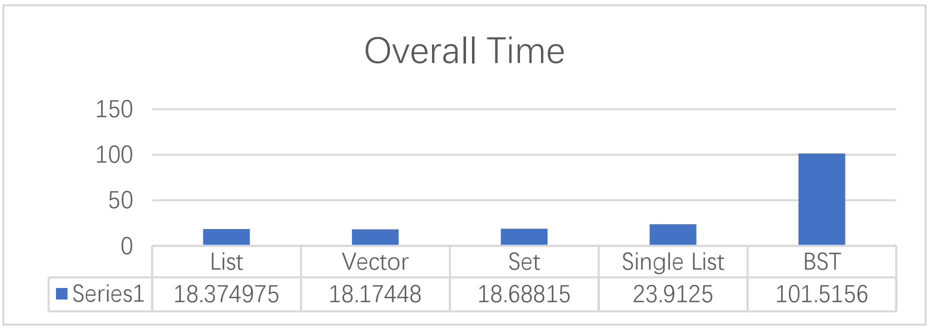
6) Summary

In this application set and BST may have least time, vector take the second location, finally list and single list takes longest time. All related reason illustrates above.

3. Data structure analysis from practical

Sometime analysis is cheap, practical analysis can examine things true power.
As mention above, it will be separate into three sections, overall time, insertion time and compare time. Input volume will have 12000, 24000 and 480000 to see how volume will influence difference data structure.

Overall time

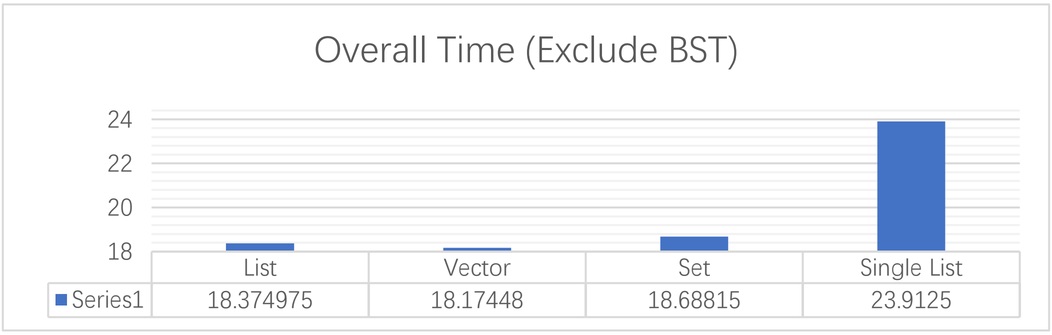


	List	Vector	Set	Single List	BST
■Series1	18.374975	18.17448	18.68815	23.9125	101.5156

Graph 1.1

From graph 1.1 we can easily find that BST overall time which is extremely more than any other data structure. As mentioned in previous Section 2.5 whole data structure implemented Leveshtein algorithm while comparing consumes much more time than any other structure, especially more than vector and list.

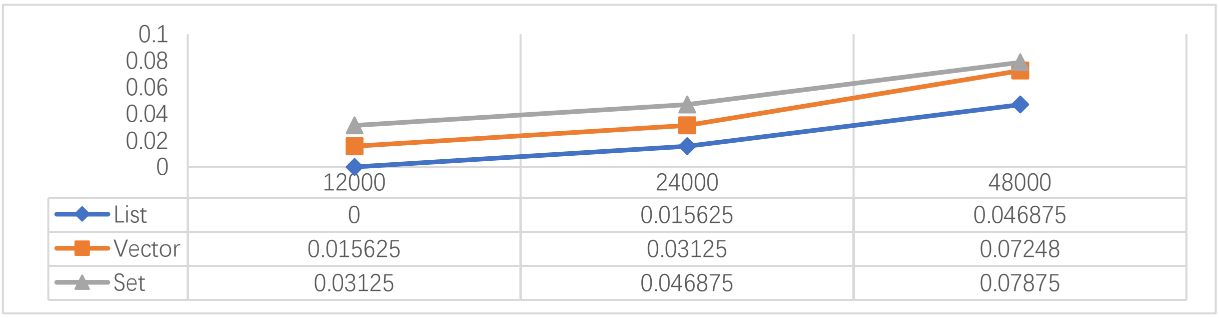
However, what is the situation excluding BST?



	List	Vector	Set	Single List
■Series1	18.374975	18.17448	18.68815	23.9125

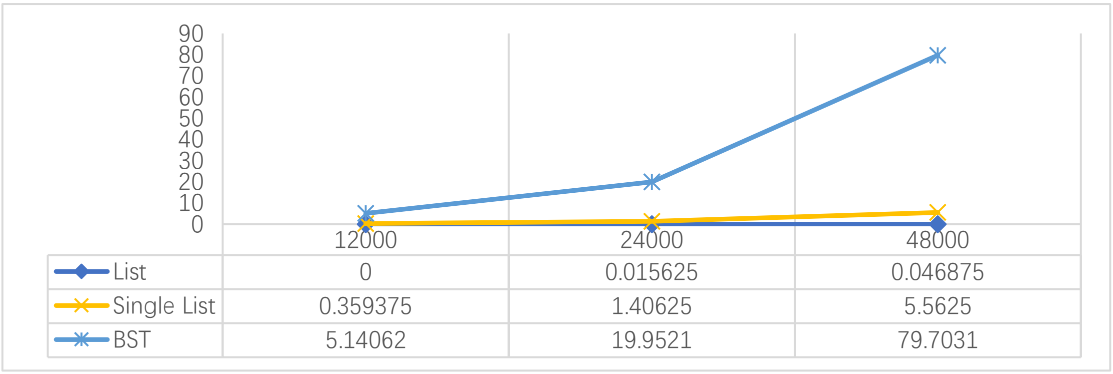
From this char situation becomes very interesting. As estimated vector uses less time because only insert word only one-time whereas access data thousands of time. Accessing speed to vector does quick. However, from theory analysis set should use the least time, and single list should have same time efficiency as STL list which is contradict to reality. Reason will be analysis in different part of implementation.

Insert time



	12000	24000	48000
List	0	0.015625	0.046875
Vector	0.015625	0.03125	0.07248
Set	0.03125	0.046875	0.07875

For List, Vector and Set, the result is just follow the prediction. List consumes least time, particularly when volume is 120000 word it even cannot calculate by system! Vector's time growth is a kind of boost as input volume growth. As we can see at first vector just consumes half time of set when input is 12000 words, however, when it comes to 48000 which nearly reaches set's time.
List's amazing insert time is mainly because list's uncontiguous memory, no resize, no relocate, no compare all of this build up list terrific inserting time.
Though there's no middle insertion, which will cost time greatly and, we still cannot ignore the time that vector resize is memory or relocate the memory in order have contagious memory. As memory growth, operations like resize and relocate require longer time. It reflects that vector does not suitable for the data that needs always change size and middle insertion.
Set, from this graph, reader may abandon this STL in this kind of string compare operation. Please do not do that. Though it always takes longest time in this char, last time's volume of insert words become four time than the first one time consuming just goes double. At the same time, List and Vector's 48000's insert time is MORE THAN four times of 12000's. This situation notifies us, we cannot ignore Set while there's huge volume of data needs to be inserted.
So, all in all we find that during insertion, List wins Vector because of memory's attribute. Whether choose Set depends on volume

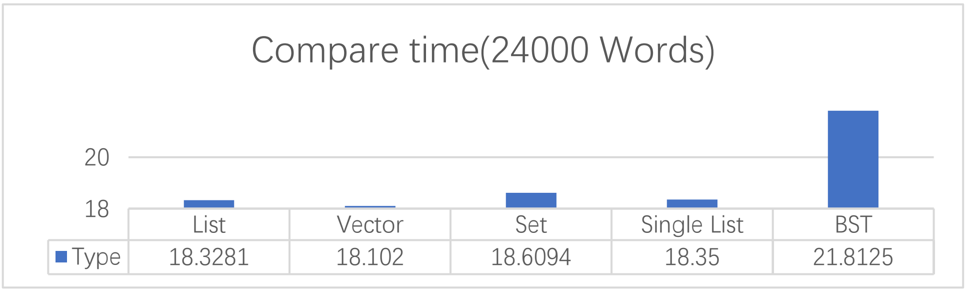


	12000	24000	48000
List	0	0.015625	0.046875
Single List	0.359375	1.40625	5.5625
BST	5.14062	19.9521	79.7031

As we can see single list has a dramatically time growth. This circumstance dramatically disobeyed estimation. When implementation examined, a critical shortage is found. Single List only has a header pointer but no end or last pointer. This means every time I insert a node. I need to start from head pointer, then iterate, finally get the end of the list, step of iteration consumes time.
Finally, when BST data comes out, it is shocking. It warns us that how the compare algorithm will affect insertion of BST. Simpler is better for BST insertion algorithm.

String compare time

Though it is string compare time, all of data structures use the same compare methods so from another side it is just related to access speed to data structure. This part is a retrieve speed competition



	List	Vector	Set	Single List	BST
■ Type	18.3281	18.102	18.6094	18.35	21.8125

As we can see from this chart, vector use least time, Single List and List are almost same speed as estimated, which is just a little slower than vector. Set takes the third place.
There's no wonder that vector's contigous memory gives application strong abilities while retriving data. Though List and single list is quick enough while accessing, it's uncontigious mememory leads them always a bit slower than vector and no thing can do to change this situation. This minor difference does not affect users who just process small bulk info, whereas, what about a software for a company that may produce TBs or even EBs level data every day? Small things do infect huge. May be just use vector instead of list can save hours for a company.

Finally, Result of set and BST compare are unexpected. From the structure aspect the target of them is find and search but they consumes longest. After analysis of application, we found that this is not only find whether in dictionary but also need all related words, which means need to iterate whole data container it is the shortage of set and BST. Furthermore, set and BST need to find correct next pointer waste some time.
When we come to talk about set. Set just uses uncontigous memory. There's no wonder that set must be slower than list while getting. This is still not enough to illustrate the circumstance in this chart, which set takes longer time than list. As mentioned in theory part. Set implemented a red black tree. Each node has a current node information, left node pointer, a right node pointer and a parent pointer, that makes looks like a complex version of list (Every brunch of a red black tree can seem as a list), more complex pointer leads to more time. Although, set has a longer iteration time than any other structru, it not means that it is not useful. When we come to search a specific value in a sequence of data, it advanced algorithm will show its power.
BST iteration time really conspicuous but much more accepateble than insert time. After comapre between BST that I implemented and Set I found some difference. Firstly, BST is not a balanced tree! Unbalanced three leads to longer time to get to another branch while the current branch are all iterated. Then, there's a IO step while iterating. When I want to get to next node. It contains:

- Ask whether there's a left or right branch,
- Ask is the next node iterated?
- If not go to that node, and insert a "iterated" status
- When a iterate is complete. I need to iterate whole tree and reset iterate status to prepare for next itereate!!!

All step do harm time efficiency, especially the last step. It makes a tree have iterate two time in one iteration operation! It is urgent to improve!

4. Things can be improved

1) For vector insertion method, a list can be used in pre-insert step. All data insert into a list first. After all data are inserted. We get the size of pre-insert list to resize target vector. But more size than list, in case of later few data insertion. We can save multiple resize and relocate memory's time.
2) For single list, besides head pointer, an end pointer is also needed. Once end pointer is added, reduce the iteration time of whole list. Dramatic fall of insertion time can be seen
3) For BST, as said above improvement of insertion compare is an urgent issue. While retrieving, a new iterate method, which do not depend on node status and do not need rest all nodes status, need to be imported.

5. Summary

STL containers as well as data structure we implemented are all can store information. Though they all can be used to same task, different structure can influence time efficiency and space efficiency greatly. For List and Single list is suitable for those need fast and random insert, like quick sort. For vector, it is useful when a sequence of data insertion time is just a few, but always access frequently. Set and BST are more useful when we find a specific point. Set do shows it power when there's tons of data. But, do not use Set and BST while it need to iterate whole structure.
From another side, we can find that STL sometime is powerful and stable enough to use. So, when we want to write a new data structure. We need to think about does this essential that can dramatically contribute to time and space efficiency? Does this worthy on cost? Furthermore, just like Single List and BST, part of the components is lost will make it consumes much more time self-implemented data structure need experiment and improvement.

6. Reference

http://www.cprogramming.com/tutorial/stl/stllist.html
http://www.cprogramming.com/tutorial/stl/vector.html
https://stackoverflow.com/questions/2209224/vector-vs-list-in-stl