Marcela Gomez

CSS 342 Program 4

# Skip List Performance Report

## Introduction

The objective of this program is to learn the construction of a new data structure, gain experience in the use of pointers and analysis of memory leaks using Valgrind in Linux. This is an invaluable skill in C++ as memory leaks can be costly to the efficiency of memory availability. Program can compile without revealing memory leak issues so proper testing in both syntax errors and memory leak are an essential part of testing your programs.

## Methods

Using the provided source code an insert, clear and remove functions were created to create

A Skip list was constructed using linked list nodes. The lowest level is filled using a search pointer to locate a value larger than our object. Once the insertion point is located, the new node is initialized and inserted to the right by reassigning pointers to the new node. We then traverse to left until an up node is located. The pointer moves the location to the next level up, searches for insertion point and inserts a newly created node into the level above the lowest level. It then repeats to fill the list where there is available up pointer, up to one level short of the top level which remains empty except for the tail and header nodes.

Using the provided driver output was checked to verify proper construction of the skip list and its multiple levels. The statistics driver is compiled to check the number of operations it takes each kind of list to perform each task. A comparison follows.

## Results

Using the output from compilation of the statistics.cpp file to generate the cost of performing a find and insert function from four types of sort lists. We can see in Figure 1. that when using 10 numbers from the 1000 randomly integer array the cost or number of operations required to complete the task is very high for the skip list. The best performer at 10 numbers is the transposition list.

**Figure 1: Number of Operations to Find/Insert 10 Random Items**

On the second try I used 35 number from the randomly generated array suddenly dlist drastically increased in number of operations. Figure 2 shows skip list changes very little. Mtflist shows only a slight increase from the cost of generating operation counts from 10 numbers.

**Figure 2: Number of Operations to Find/Insert 35 Random Items**

Progressing the size to 50 we see the results in Figure 3. Dlist is consistently performing at a higher cost of operations to complete the find and insert task over the other three. Mtflist continues to outperform all other methods. Skip list and transposition list continues to improve at a steady rate with increase in number of integers to perform operations upon.

**Figure 3: Number of Operations to Find/Insert 50 Random Items**

At size 75, dlist is far surpassing the other three list types at 41,308 operations needed to finish all tasks.

All three other lists are steadily increasing, while dlist Is on a steady rise. Let’s move onto observe the behavior of each list over the increase of all sized in comparison to each other.

**Figure 4: Number of Operations to Find/Insert 75 Random Items**

Over the increase in size of numbers from 10 to 75 to perform operations on, mtflist stays under 4000 operations. Skip list starts at 8812 and increases to 12,266. Dlist started off well and shows erratic performance with increase in size of number to handle.

**Figure 5: Cost Comparison of Operations for Four List Types: d/mtf/trans/skip Lists.**

## Memory Leak Analysis and Resolution:

Using the valgrind with debugger -g to produce line specific error report I was able to get better insight into the state of the code.

**Figure 6. Val grind Memory Leak Debug Line Commands**

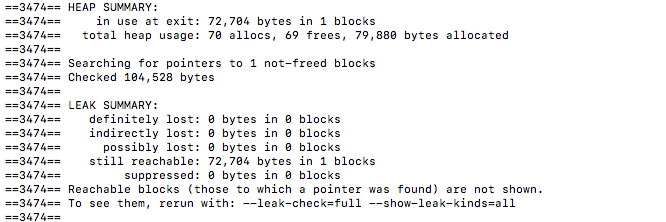
**g++ statistics.cpp -g**

**valgrind – leak-check=full -v ./a.out 10**

From researching on the internet. I concluded that under definitely lost and indirectly lost there are 0 bytes lost. The reachable bytes were described as a reflection of the standard library associated with the program.

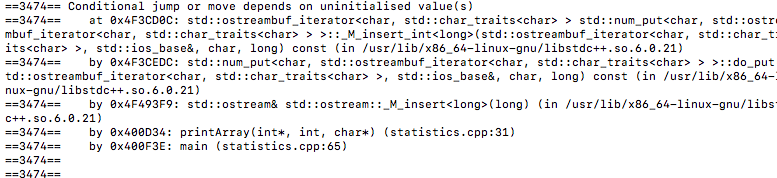
In order to get to this stage two step had to be taken to assure that definitely/indirectly lost bytes was reduced to 0. First of all, I had to take a slow step by step review of the pointers in the insert function. I had mis assigned the pointer of one node prematurely resulting in the node pointing to itself. Delete statements were added at the end of statistics.cpp to remove array items, pattern, frequency and sequence.

**Figure 7 valgrind Memory Leak Debug Report**



Once that was repaired there was still some loss of memory stemming from error of randomly generated memory allocation from the cout function on like 31 and 64 as per the valgrind report. Researching this on stack over flow the error was determined to stem from a weakness in valgrind library as the report was run across all four list types used to generate the statistics file.

**Figure 8 cout memory error report stemming from statistics.cpp file.**



## Discussion

This was an interesting project of substantial challenge to construct this data type and proof for errors. Despite the challenge it was an appreciable process in that in cause me to engage more with Linux and error debugging in C++. The outcome was understanding ways to test and use data types depending on their ability to perform with different amount of data.

## Conclusion

While skip list data type proved to me an interesting data type, mtflist outperformed all other lists in this experiment. Transposition list was a close follow up to mtflist performance.