Cameron Keith

CSE 7350 – McFearin

11 October 2018

Conference Session Scheduling Project

PART 1

For this project I decided to build and run the code with Java on a Macintosh computer. The technical specs for the Java VM and computer hardware itself are 4GB for the Java Heap Space, 16GB for the computer’s RAM, and a 2.8Ghz Intel i7 processor for the computation. For the Personal Distribution choice, I went with recreating the Normal Distribution Curve and I achieved this distribution by taking the average of three random numbers chosen from the set of total possible sessions. I achieved the Uniform Distribution by generating a random number between 1 and the total session count. The Skewed Distribution was creating by taking the lesser of two random numbers generated from 1: total session count. Finally, the Two-Tiered Distribution was generated by generating the result of a coin toss and depending on the result of the coin toss determined the upper and lower bounds for the Random Number generation for that session attempt. With each of the four distribution attempts I had an additional check before generating the next session number to ensure that the generated session number was not a session that the attendee was already attending thus not allowing any duplicates to be chosen. For the Histograms representing the various distributions; see Appendix C.

For Generating the K Sessions for S attendees, the space complexity is Θ(S\*K) where the total space required will always be the Number of attendees multiplied by how many sessions they are attending. For the time Complexity it bounded by Ω(S\*K) on average and O(N!) at the worst. This is because for values of K less than N/2 the time to generate the total number of random numbers for each attendee will be approximately proportional to S\*K. As K approaches the value of N for large values of N the number generation pattern I have chosen just has you picking another random number. As you pick more and more numbers eventually you are having to attempt an increasingly large numbers of times to finally pick the ones you are missing leading to an O(N!) upper bound.

*Chart contains Various input and runtime values for random number generation*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| N | S | K | Avg Time to Complete (Seconds) | Difference to Previous | What Changed from Previous? |
| 5000 | 12000 | 4900 | 44.46 | 13.86486486 | 4.9 \*Base |
| 5000 | 12000 | 4500 | 27.48 | 8.56964657 | 4.5 \*Base |
| 5000 | 12000 | 1000 | 3.207 | Base | Base |
| 10000 | 12000 | 100 | 0.226 | 0.07 | K/10 |
| 10000 | 12000 | 200 | 0.421 | 1.87 | 2K |
| 10000 | 12000 | 400 | 0.918 | 2.18 | 2K |
| 10000 | 24000 | 400 | 2.647 | 2.88 | 2S |
| 5000 | 24000 | 400 | 2.480 | 0.94 | .5N |

For removing the duplicates, I used the following methods.

The First method involved the creation of an NxN array and used the resulting matrix to encode if the (X, Y) location within the matrix has a Conflict between X and Y. After passing through each Attendee and what sessions they are attending I then go through the Matrix array and for all values of (X, Y) = 1 I append a new Node to a Linked List that contains the pair (X, Y) as a Session Conflict object.

The second method I employed to remove Duplicates I add to a Sorted TreeSet and if I reach a Conflict node within the Tree and I still have not found a node that is equivalent to the node I want to add I then add a new node. If in the traversal I find a node that is equivalent, then I know that this Conflict is already found within the tree and I move on to the next Conflict pair.

For the asymptomatic bounds of the space required to remove duplicates is O(N2) for all Distributions. It is bounded by O(N2) because regardless of how many attendees you have or how many of the N sessions they attend the total number of unique session conflicts is bounded by N­2 as that represents each Session having a conflict with every other session.

For All Distributions the Lower bound for the Space required to store the Unique Session conflicts is Ω(S\*K2) because each person who attends K sessions creates K2 conflicts. Thus, lower bound for Space required would then be dependent on how many people are in attendance (S) as well as how many sessions they are attending (K).

For the Time complexity the upper bound is O(S\*(N/K)2). The time necessary to compute the Number of Unique sessions grows linearly with the Number of attendees S as each new attendee will add the same about of time to Process. As for N and K, as K approaches N the number of Unique conflicts for the group approaches the maximum number of possible conflicts N2 so as N and K begin to approach the same value the time necessary will begin to level out.

PART 2

[WalkThrough]

The Asymptotic space required for the Random and In Order Coloring Algorithms is Θ(M+N). This is because the storage for loading the sessions is of size N and each session has its own collision list that when added is equivalent to an array of size M, the color of a given session is also stored in the session object giving N the value 2N but for asymptotic purposes can be reduced to a size of N. Thus the space necessary to compute the coloring of a given set of sessions can be computed with M+N space complexity.

The Asymptotic time required for the Random and In Order Coloring array is upper bounded by O(M) this is because the implementation steps through the storage structure incrementally across the sessions and a list of which sessions they collide with. If you were to compute the total sum of the sessions collision lists you would get the value M data accesses which gives the asymptotic timing of O(M).

*The following chart contains the timing of various combinations of N,S,K and the resultant colours*

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | Seconds | | Colours | |  |  |
|  | N | S | K |  | IO | RO | IO | RO |  | M |
| Uniform | 6000 | 80 | 600 |  | 0.150655 | 0.141911 | 3788 | 3696 |  | 19618056 |
| Uniform | 5000 | 2000 | 100 |  | 0.119655 | 0.074873 | 3041 | 3052 |  | 13456338 |
| Skewed | 2500 | 1000 | 250 |  | 0.047224 | 0.045902 | 2198 | 2305 |  | 5622034 |
| Personal | 2000 | 200 | 200 |  | 0.017788 | 0.025063 | 1239 | 1327 |  | 2093794 |
| Tiered | 1000 | 80 | 480 |  | 0.013095 | 0.015488 | 1000 | 1000 |  | 999000 |

Appendix A:

Example Output File 1: Inputs - 2000 200 200 PERSONAL

N  
2000  
S  
200  
K  
200  
M  
2110836  
T  
7960000  
DIST  
PERSONAL

Example Output File 2: Inputs - 2500 1000 250 SKEWED

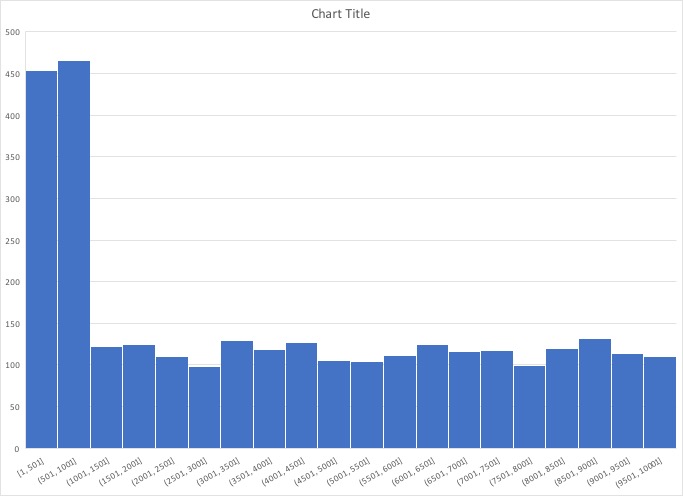
N  
2500  
S  
1000  
K  
250  
M  
5621360  
T  
62250000  
DIST  
SKEWED

Example Output File 3: Inputs - 5000 2000 100 UNIFORM

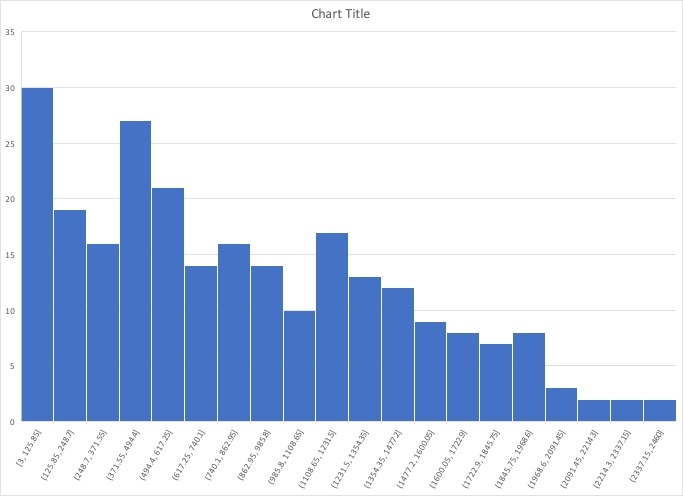
N  
5000  
S  
2000  
K  
100  
M  
13673346  
T  
19800000  
DIST  
UNIFORM

Appendix C:

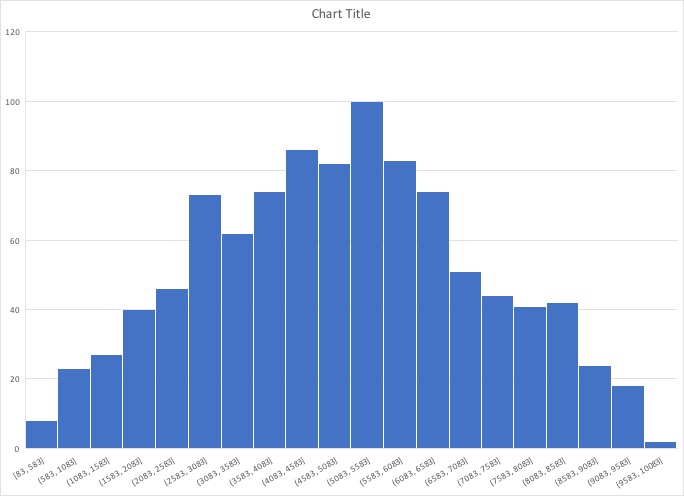
Tiered Histogram



Skewed



Personal



Uniform

