Valgrind and Profiler Report

1 Valgrind

1.1 Part 1

We ran our program and found a few definite and indirect losses in our program (loss of about 150 KBs). We had made sure that all our classes had destructors, but we found that adjacency matrix and students vector did not clear on their own. So we cleared them manually before program terminates and ran valgrind again. There were no definite or indirect losses now. The valgrind output is attached as "valgrind.txt".

1.2 Part 2

In part 2, we made sure that there were no possible memory leaks while writing code. Hence there is no definite or indirect loss. The valgrind output is attached as "valgrind2.txt".

2 Profiler

We have used the GNU Profiler to profile both our generator and analyzer (the output reports of both of them are attached as "profiler1.txt" and "profiler2.txt" respectively.

2.1 Part 1

The following is the flat profile showing major contributors of time, when we ran the simulation for 14 years:

time %	time taken	function name
4.86	0.32	void std::_Construct<std::pair, std::pair const&>
4.86	0.32	bool std::operator< <std::string, int>
3.57	0.24	bool std::operator< <char, char_traits<char>, allocator<char> >
3.49	0.23	generateRequest(int)

Most of the runtime was spent on generating friend request, which is reflected in profiler report. Since we stored each person data in form of pair(student id, university name) during friend generation, it took most of the running time. Also addition to adjacency list were made during friend request generation, which is also reflected.

2.2 Part 2

The analyzer that we made ran graph algorithms. So, a lot of allocation is done for storing nodes and edges. We used stl vectors and maps for that purpose. That is also reflected in profiler report as large number of allocators and destroyers to stl structures calls.