Internship Report

Murat Bolu

September 1, 2024

My internship at TÜBİTAK BİLGEM İLTAREN was from 1–29 July 2024 at the Ümitköy campus. My department was system administration software, and I completed the tasks in C++. My progress was monitored primarily by Emre Balcı and Murat Özdemiray.

1 File Organization

2 Static Data Structures

I started the internship writing statically allocated data structures from scratch. Even though the C++ standard library provides some statically allocated data structures like `std::array`, my mentor requested I write them by myself and avoid using the standard library. Statically-allocated data structures are preferred in low-level applications for several reasons. First, they do not use `malloc` or `new`. These methods require an operating system or an equivalent program that will provide the necessary memory. These kinds of memory managers are not present in embedded applications and must be avoided. Second, they do not utilize the stack. While the stack may feel like an infinite memory source, stack overflow poses a significant risk for such applications. Statically allocated data structures allocate their memory at once while the program starts, and the stack is not utilized except for small memory allocations like `int`'s and pointers.

3 Traveling cities problem

In the first internship task, I was provided an XLS file containing the distances between each pair of provinces in Turkey. I exported that XLS file to a CSV file and did some very light preprocessing by hand. I removed the first two lines containing the title text and junk information. Only 81 lines containing the names of the provinces and the distances from that province, separated by semicolons, were left. I wrote a parser that imported that text file into a data structure in memory. Finally, I had an array of arrays, with size 81 times 81, containing the distances as a number.

With this data, my task was to visit as many provinces as possible, given some constraints. The distances between provinces were not considered except to apply constraints. After the constraints were applied, the problem transformed into the longest simple path problem in an undirected, unweighted graph, with the goal of visiting most vertices.

The longest simple path problem is finding the longest path in a graph without visiting any vertex more than once. It is an NP-Hard problem, and even the approximations are computationally complex. The best way to approach this problem is through heuristics. I have experimented with several heuristics and chosen to implement some of them.

In our graph, all pairs of provinces are connected through an edge. These edges have a value corresponding to the distance. Naturally, some of these edges must be invalid since the longest simple path in a connected graph is trivial. They are invalidated through three constraints. The first is the starting province, and the second and the third are X and Y values. X value represents the distance, and Y value is the error margin of that distance. For example, if X is 200 and Y is 50, only edges with values between 150 and 250 are valid.

4 Bus scheduling problem

```cpp
#include "exec/BusAnalyzer.tpp"

#include <cstdlib>
#include <optional>

constexpr static unsigned samplingPeriod { 1 };

int main(int argc, const char* argv[])
{
    std::optional<BusAnalyzer> ba { BusAnalyzer::create(argc, argv,
   samplingPeriod) };
    if (!ba.has_value())
    {
        return EXIT_FAILURE;
    }
    ba->getSamples();
    ba->parseSamples();
    ba->extractPeriods();
    ba->printResult();
    return 0;
}
```

Listing 1: `file.cpp`