Analysis of Algorithm Complexities

Microsoft 2 (team name i think)

March 13, 2024

1 Introduction

We analyze big O of college.

2 Code Snippets

Below are the relevant code snippets extracted from the application:

2.1 CSV Parsing Function (csv_to_df)

```
void MainWindow::csv_to_df(string path, QMap<QString, QMap<QString,
     double>> &dataframe)
{
    ifstream csv(path);

    if (!csv) {
        cout << "Could not open file! :(" << endl;
    }
    char ch;
    string buffer;
    unsigned quotes = 0, count = 0;
    string row, col, val;

    csv.ignore(1000, '\n');
    while ((ch = csv.get()) != EOF) {
        if (ch == ',' && quotes % 2 == 0) {
            switch (count % 2) {
            case 0:
                row.assign(buffer);
                break;
            case 1:
                col.assign(buffer);
                break;
            }
            count += 1;
            buffer.assign("");
        }

        else if (ch == '\n') {
            val.assign(buffer);
```

```
            buffer.assign("");
            // cout << row << " - " << col << " - " << val << endl;
            dataframe[QString::fromStdString(row)][QString::
    fromStdString(col)] = std::stof(val);
        }

        else if (ch == '\"') {
            quotes += 1;
        }

        else {
            buffer += ch;
        }
    }
}
```

Listing 1: CSV Parsing Function

Let n be the number of characters stored in the CSV file. Since the function reads through each character of the file individually, deciding where to assign them based on certain delimiter values, the time complexity of the function is $O(n)$.

2.2 Shortest Path Finding Function (find_shortest_path)

```
QVector<College> *MainWindow::find_shortest_path(QString location,
    int n, QVector<College> *trip)
{
    if (trip == nullptr) {
        trip = new QVector<College>{};
    }
    trip->push_back(location);

    if (n > 1) {
        int min = 100000;
        QString next;
        for (auto i = TripColleges.begin(); i != TripColleges.end()
    ; i++) {
            const auto eqCollegeName = [i](College &s) { return s.
    name() == i->name(); };
            if (distanceMap[location][i->name()] < min
                && std::find_if(trip->begin(), trip->end(),
    eqCollegeName) == trip->end()) {
                min = distanceMap[location][i->name()];
                next = i->name();
            }
        }
        return find_shortest_path(next, n - 1, trip);
    } else {
        return trip;
    }
}
```

Listing 2: Shortest Path Finding Function

Let n be the number of colleges that the function needs to path between. The function calls itself recursively for every college in the list. Each recursion runs a for loop with a range of n, inside of which there is the `std::find_if` function, which also has a time complexity of n. Therefore, the time complexity is $O(n^3)$.

2.3 Adding a College to the Trip (on_button_addToTrip_clicked)

```
void MainWindow::on_button_addToTrip_clicked(bool checked)
{
    QString text = "";
    currentCollege->toggleInTrip(checked);

    if (checked)
        TripColleges.append(*currentCollege);
    else {
        for (int i = 0; i < TripColleges.length(); i++) {
            if (TripColleges[i].name() == currentCollege->name()) {
                TripColleges.remove(i);
                break;
            }
        }
    }

    TripColleges = *find_shortest_path(TripColleges[0].name(),
    TripColleges.length());

    ui->label_tripColleges->clear();

    int totalDistance = 0;
    for (int i = 0; i < TripColleges.length() - 1; i++) {
        totalDistance += distanceMap[TripColleges[i].name()][
    TripColleges[i + 1].name()];
        text += TripColleges[i].name() + " > "
                + QString::number(distanceMap[TripColleges[i].name
    ()][TripColleges[i + 1].name()])
                + "mi > ";
    }

    if (TripColleges.length() != 0)
        text += TripColleges[TripColleges.length() - 1].name();
    ui->label_tripColleges->setText(text);
    ui->label_totalDistance->setText("Total Distance: " + QString::
    number(totalDistance));
}
```

Listing 3: Adding a College to the Trip

Let n be the number of colleges that the function needs to add to the list. This function runs a for loop across the entire list of colleges, inside of which it retrieves values from a map. Because map retrievals have a time complexity of $\log(n)$, the time complexity of the function is $O(n \log(n))$.