tags: `Report`

# Real-Life Applications of Computational Algorithms - HW3

309551130 謝柏威

## file structure

```
.
├── makefile
├── data
│   ├── input_10.txt
│   ├── input_200.txt
│   ├── output_10.txt
│   └── output_200.txt
├── doc
│   ├── hw3_description.pdf
│   └── report.pdf
└── src
    └── main.cpp
```

# Usage

- `makefile` is provided.
- All the source code can be compiled and run on `linux{id}.cs.nctu.edu.tw` by simply typing `make`. The resulting binary will be generated named `viterbi`.

```
$ make
g++ -std=c++17 -Ofast -Wall -Wextra -Wshadow -o viterbi src/main.cpp
./viterbi
Usage: ./viterbi infile outfile

$ la viterbi
-rwxr-xr-x 1 pwhsieh gcs 38K May 30 17:24 viterbi
```

# Implementation

1. main function
2. HW3 struct

3. viterbi algorithm

# 1. main function

- In main function, the input file is read by the `ifstream`, which is given by the first argument.
- All the parameters given in the spec is wrapped inside the `HW3` struct.

```
auto is = ifstream(argv[1]);
auto hw = HW3{};
is >> hw;

auto os = ofstream(argv[2]);
hw.result = Viterbi(hw.param()).predict(hw.observe);
os << hw;
```

# 2. HW3 struct

- As previously mensioned, all the parameters mensioned in spec is in this structure.
- Here's the brief overview of the structure.

```
struct HW3 {
    HW3() {
        start_prob = vector{0.5, 0.25, 0.25};

        transition = vector {
            vector{0.8, 0.15, 0.05},
            vector{0.2, 0.5, 0.3},
            vector{0.2, 0.2, 0.6}
        };

        emission = vector {
            vector{0.9, 0.1},
            vector{0.7, 0.3},
            vector{0.2, 0.8}
        };

        weather = {"sunny"s, "foggy"s, "rainy"s};
    }

    vector<double> start_prob;
    vector<vector<double>> transition;
    vector<vector<double>> emission;
    vector<string> weather;
};
```

# 3. viterbi algorithm

- This is the main part of the algorithm.
- `Viterbi` is a dynamic programming algorithm. In each iteration, the algorithm will pick the state with the largest probability as it's predecessor.
- In the end, the algorithm will get the largest probability for the given observered data.
- The prediction can be obtained by mentaining the `path` variable.

```
auto predict(vector<int> &observe) {
    auto dp = start_prob;

    // initial setup
    for (int i = 0; i < n_tran; i++) {
        dp[i] += emission[i][observe.front()];
    }

    // viterbi
    auto path = vector(n_tran, vector(observe.size(), -1));
    for (size_t i = 1; i < observe.size(); i++) {
        auto tmp = vector(n_tran, -DBL_MAX);
        for (int cur = 0; cur < n_tran; cur++) {
            for (int pre = 0; pre < n_tran; pre++) {
                if (auto M = dp[pre] + transition[pre][cur]; M > tmp[cur]) {
                    tmp[cur] = M;
                    path[cur][i] = pre;
                }
            }
            tmp[cur] += emission[cur][observe[i]];
        }
        dp.swap(tmp);
    }

    // generate the path
    auto result = vector<int>(observe.size());
    int val = max_element(dp.begin(), dp.end()) - dp.begin();
    for (int i = observe.size() - 1; i >= 0; i--) {
        result[i] = val;
        val = path[val][i];
    }

    return result;
}
```

# Results

- It seems like the output is same as the given answer. (except for missing new line in the end)

```
$ ./viterbi data/input_10.txt ans
$ diff data/output_10.txt ans
12d11
```

```
<

$ ./viterbi data/input_200.txt ans
$ diff data/output_200.txt ans
202d201
<
```

# Discussion

- The assignment helps me a lot on getting an insight into the concept.
- After implementing the whole algorithm from scratch, I think I have a pretty good understanding on `HMM` and `viterbi` algorithm.