Weather Warning and Station Optimizer

CMPSC 463 (section 1) GitHub Avik Bhuiyan & Allen Chea

Project Description

This project's goal was to implement an early weather warning system with a weather station coverage optimizer. It includes 2 main features:

1. *Weather Warning Alert*
    a. The weather warning features gathers data from U.S. National Weather Service live, analyzes it, and returns the corresponding alert (low, moderate, high), along with displaying measurements such as humidity and temperature.
2. *Weather Station Optimizer*
    a. The station optimizer tool lets users create weather stations and can choose which regions they cover, meaning it can do weather measurements there. It finds the best k stations to maximize the total area coverage by utilizing a *greedy selection* algorithm.

Significance

The significance of this project is not new technology but demonstrates new skills we learned throughout the course and also could be used for weather emergencies. The weather alert system feature can be used by the public during events such as hurricanes, storms, or floods, or even by emergency response teams to know when to send out their mass public warnings.

The station optimizer can be used by weather agencies to determine where to install their weather measuring or weather monitoring systems/stations to maximize the area and save costs/resources.

Code Structure

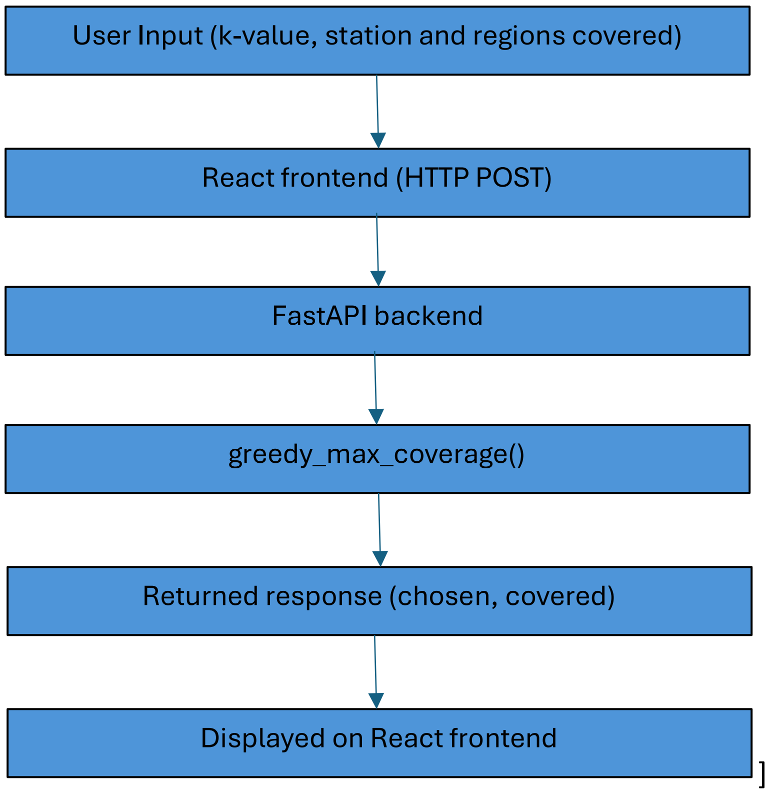
This project is ran using a FastAPI backend and React frontend. The root project directory contains a folder that distinguishes the front and backend.

```
Weather-Crisis/

Backend/

main.py                 // python script to run FastAPI

Frontend/
```

Src/

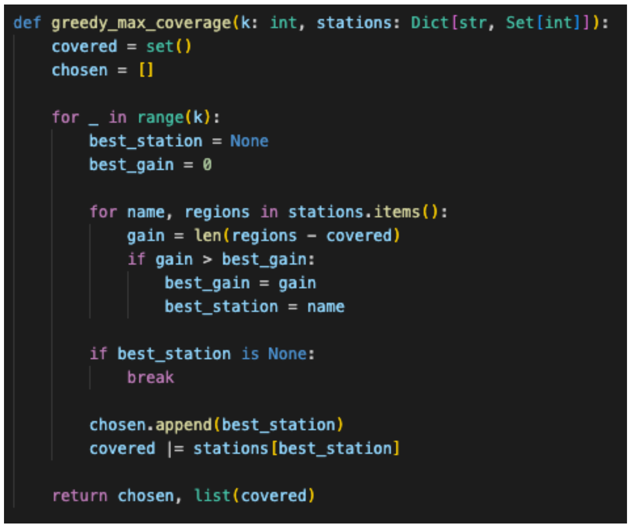
App.js // main component containing both features

To run this project, simply use the launch configuration which runs the FastAPI backend + React front-end and automatically open the webpage.


User Input (k-value, station and regions covered)
React frontend (HTTP POST)
FastAPI backend
greedy_max_coverage()
Returned response (chosen, covered)
Displayed on React frontend
]

Description of Algorithms

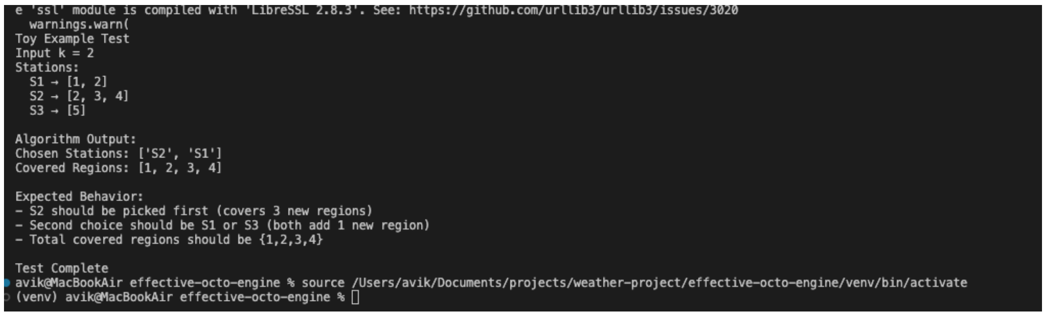
The weather station coverage optimizer uses a greedy maximum coverage algorithm. This algorithm selects k stations that cover most areas in regions together. It starts with an empty set, repeats k times, and then returns the chosen stations and regions covered. This is an implementation of a maximum coverage problem.



```python
def greedy_max_coverage(k: int, stations: Dict[str, Set[int]]):
    covered = set()
    chosen = []

    for _ in range(k):
        best_station = None
        best_gain = 0

        for name, regions in stations.items():
            gain = len(regions - covered)
            if gain > best_gain:
                best_gain = gain
                best_station = name

        if best_station is None:
            break

        chosen.append(best_station)
        covered |= stations[best_station]

    return chosen, list(covered)
```

Verification of Algorithms

To test the project's algorithms, we reated a testing suite for the algorithms using a Python script. It had a control. Dataset with expected outcomes and resulted in no errors.



```
e 'ssl' module is compiled with 'LibreSSL 2.8.3'. See: https://github.com/urllib3/urllib3/issues/3020
  warnings.warn(
Toy Example Test
Input k = 2
Stations:
  S1 → [1, 2]
  S2 → [2, 3, 4]
  S3 → [5]

Algorithm Output:
Chosen Stations: ['S2', 'S1']
Covered Regions: [1, 2, 3, 4]

Expected Behavior:
- S2 should be picked first (covers 3 new regions)
- Second choice should be S1 or S3 (both add 1 new region)
- Total covered regions should be {1,2,3,4}

Test Complete
avik@MacBookAir effective-octo-engine % source /Users/avik/Documents/projects/weather-project/effective-octo-engine/venv/bin/activate
(venv) avik@MacBookAir effective-octo-engine % 
```

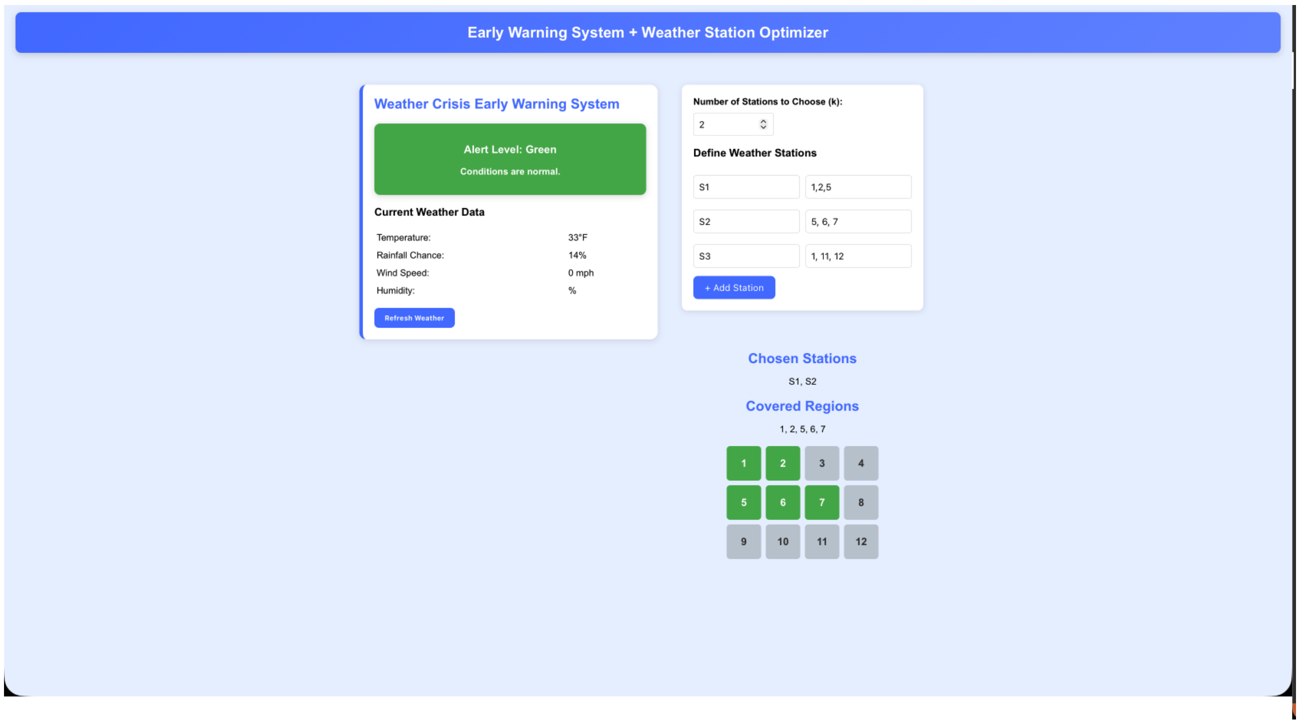
This testing made sure that our greedy algorithm behaved as expected.

Functions

The project used a few functions including:

1. parse_regions()
    a. Verified that it correctly converted the user input strings into integer lists such as "1, 2, 5" to [1, 2, 5].
2. compute_cover(stations, k)
    a. Verified input validation.
    b. Ensured the function returns a correct list of chosen station names and a combined set of all covered regions
    c. Checked that the function never mutates the original station list.
3. API Call Verification
    a. Tested both valid and invalid requests using HTTP clients through the localhost.

Results and Analysis



Early Warning System + Weather Station Optimizer

Weather Crisis Early Warning System

Alert Level: Green

Conditions are normal.

Current Weather Data

Temperature:	33°F
Rainfall Chance:	14%
Wind Speed:	0 mph
Humidity:	%

Refresh Weather

Number of Stations to Choose (k):

2

Define Weather Stations

S1	1,2,5
S2	5, 6, 7
S3	1, 11, 12

+ Add Station

Chosen Stations

S1, S2

Covered Regions

1, 2, 5, 6, 7

1	2	3	4
5	6	7	8
9	10	11	12

The front-end features a clean, slick UI with the 2 core features side by side. The left side is the weather warning system displaying some current weather conditions and the current alert status, which is currently normal since nothing triggers the alert.

The right side displays inputs for the k-value, and also for the weather stations. It also displays the region grid and highlights the maximum coverage using the greedy algorithm.

Conclusion

Overall, this project demonstrates how algorithms can be used in the real world and learning outcomes from the course. This project could be improved by fetching the weather automatically and sending out automated alerts, along with a more in-depth station coverage optimizer. However, the project still successfully implements greedy methodologies to maximize region coverage with the weather stations.