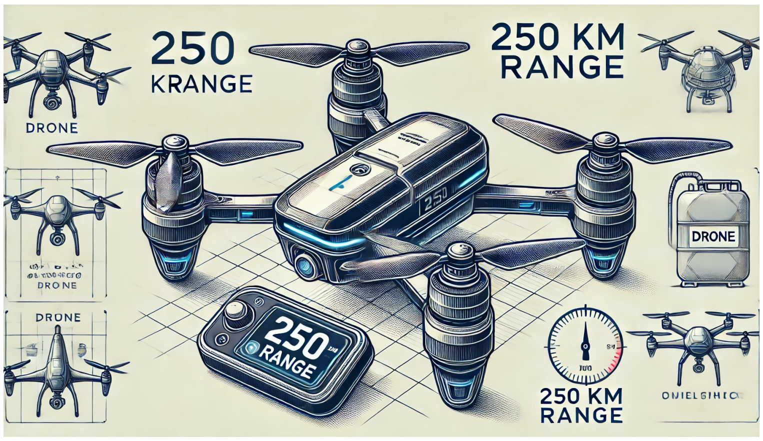
BBM203 Data Structures Laboratory PA3
Fall 2024
Instructors: Assoc. Prof. Dr. Adnan Özsoy, Asst. Prof. Dr. Engin Demir, Assoc. Prof. Dr. Hacer Yalım Keleş
TA's: M.Aslı Taşgetiren, **S.Meryem Taşyürek**, Asst.Prof.Dr. Selma Dilek

Drone ALPHA Journey Through Türkiye

Due date: Tuesday, 01-12-2024, 11:59 PM.

Wisdom of the Path: Stack Your Knowledge, Queue Your Moves!

In this assignment, you are part of an elite logistics team working with the Turkish Provincial Logistics Authority (TPLA). They are tasked with programming and guiding a fleet of autonomous drones to complete critical supply deliveries across Türkiye. But the journey isn't as simple as moving from point A to point B. Each province has unique connections, and the drones have limited fuel reserves that only allow them to travel specific distances between stops. You will make use of Stack and Queue data structures to create an efficient, algorithmic route-finding solution.



In the future, the TPLA has adopted cutting-edge drone technology to handle provincial logistics. Each drone is designed to autonomously navigate a set of provinces while maximizing coverage (the number of provinces visited) under specific constraints, including limited fuel capacity. As a planner, you must develop a path that allows the drones to cover the maximum number of cities while avoiding revisits and dead-ends. Their ultimate goal? To navigate through the provinces of Türkiye in the most efficient way possible!

Mission Briefing: Attention, Logistics Planners!

Welcome to TPLA's Strategic Drone Fleet. Today, you will design routes for our autonomous drones to ensure maximum provincial coverage across Türkiye. These drones are fitted with a fuel tank allowing them to only travel a specific distance. To keep our drones in operation, they need to refuel every 250 kilometers. You must program each drone to navigate through our provinces while visiting the maximum number of locations without getting stranded or revisiting old paths. To do this, you'll be equipped with Stack and Queue structures to manage route decisions and backtracking, ensuring an optimal journey for our fleet!

1 Implementing Stack and Queue Structures

First Mission: Drone ALPHA Launches from Ankara!

Drone ALPHA, our first drone in the fleet, launches from the capital, Ankara. With limited fuel and limited memory it must navigate wisely through Türkiye's provinces to cover as much ground as possible. Its mission is to explore the maximum number of provinces without retracing its steps or getting stranded. To achieve this, Drone ALPHA will use two critical tools: a Stack for backtracking through previously visited provinces and a Queue to manage potential next steps in the journey.

The drones are equipped with limited memory and so it is wise to use static, array-based data structures to control the amount of memory allocated. This choice helps ensure efficiency and prevents memory overflow as Drone ALPHA tracks its journey. Each structure will play a specific role:

- Stack (Backtracking): Helps Drone ALPHA manage its route history by allowing it to retrace steps if it encounters a "dead-end". This structure operates on a Last In, First Out (LIFO) basis, so it's perfect for storing a path that might need to be reversed.
- Queue (Route Management): Organizes potential next moves in First In, First Out (FIFO) order. By processing provinces in the order they're added, the queue ensures that Drone ALPHA explores all reachable areas efficiently.

1.1 Implementing the Stack for Backtracking

The Stack structure will help Drone ALPHA manage its journey by keeping track of the route history. Each time Drone ALPHA moves forward to a new province, it will push that province onto the stack. If it encounters a "dead-end" (where no further provinces are reachable or it hits the boundary of its fuel range), the stack will allow it to pop provinces and backtrack to explore alternate routes. Think of the stack as Drone ALPHA's "memory of recent moves." The LIFO (Last-In, First-Out) nature of a stack allows the drone to "retrace" its steps by returning to the most recent province in the journey.

Implementation Tips: Use a static array-based stack to ensure memory efficiency. Array-based structures are simple and compact, which is crucial for the drone's limited memory. Define a maximum size for the stack considering the problem at hand. Each time Drone ALPHA moves to a new province, add (push) that province to the stack. When backtracking is needed, remove (pop) the most recent province from the stack and return to the previous one until a new route opens up.

Stack Operations: Track the history of the route, allowing backtracking as necessary. Implement push, pop, and top operations to support Last-In, First-Out (LIFO) access for efficient backtracking.

1.2 Implementing the Queue for Route Management

The Queue will help Drone ALPHA organize potential next steps based on neighboring provinces. Unlike the Stack, which allows backtracking, the Queue will manage the ongoing list of provinces that Drone ALPHA plans to visit. Each time Drone ALPHA reaches a province, it will enqueue nearby, unvisited provinces, ensuring it explores every possible route in First In, First Out (FIFO) order.

Implementation Tips: Use a static array-based circular queue for efficient exploration management. A circular queue allows Drone ALPHA to handle the exploration of neighboring provinces in a First-Define a maximum size for the queue considering the problem at hand. This approach provides sufficient capacity for exploration needs while maintaining simplicity and efficiency. For each province, enqueue neighboring provinces that haven't been visited yet and are within range. This circular queue will enable Drone ALPHA to manage exploration options effectively, visiting each neighboring province in the order they're added. Once a visit is completed, the next province can be dequeued, allowing the journey to proceed smoothly.

Queue Operations: Manage provinces Drone ALPHA should consider next in the route. Implement enqueue, dequeue, and peek functions to add and remove provinces in a First-In, First-Out (FIFO) manner.

2 Setting Up the Provincial Map and Constraints

Mission Context

Drone ALPHA's journey begins in Ankara, but navigating across Türkiye's provinces is no simple task. Each province is connected to others by specific distances, and the drone can only travel a limited distance on each leg. To make the journey as efficient as possible, you must set up a map that includes distance data between provinces, choose a starting point, and apply constraints. These constraints will ensure Drone ALPHA uses its limited fuel wisely, maximizing the number of provinces visited without exceeding its travel range.

2.1 Step 1: Setting Up the Provincial Map

The provincial map should be represented by a 2D matrix (distance table), where each entry at position (i, j) represents the distance between Province i and Province j.

Why this setup? This matrix allows Drone ALPHA to calculate which provinces are reachable from each starting point, based on the defined travel range.

- You have been provided with an Excel and corresponding CSV file that includes distance information between provinces of Türkiye.
- In the 2D array you'll create, distance[i][j] will represent the distance between Province i and Province j.
- Purpose: This map will allow Drone ALPHA to assess which provinces are reachable from any given starting point.
- **Loading Data from the CSV File:** You'll create a function called *loadDistanceData* that reads the CSV file and stores the data in a 2D array, where each entry corresponds to the distance between two provinces. Ensure each distance is stored as an integer in the array.

2.2 Step 2: Setting Up the Travel Constraints

Travel constraints are essential for simulating the drone's fuel limits and route restrictions, adding complexity to the journey. You will implement these constraints that dictate when the drone must backtrack or queue potential new routes.

Defining the Distance Constraint: Each move by Drone ALPHA has a maximum allowed travel distance of 250 km. This constraint represents the drone's fuel capacity and limits which provinces are reachable from each point. The drone must refuel every 250 kilometers to continue its journey.

- Goal: Set the distance limit. Only provinces with a distance less than or equal to 250 km from the current location are considered for the next step.
- Implement the function *isWithinRange*: This function checks if the distance between two provinces is within the allowed range. It returns true if the distance is less than maxDistance, and false otherwise.
- Starting Point: Set Ankara as the starting province (province index 5), but allow this to be configurable if desired.

Priority Provinces: Certain provinces are marked as high-priority locations that the drone should aim to visit as part of the journey. These priority provinces represent strategic areas that **must be reached** for the mission to be successful.

- Goal: Keep track of the priority provinces visited. Use an array to store these provinces and check if the current province is in the priority list during exploration.
- Implement the function *loadPriorityProvinces*: Load the priority provinces from the given input file and store them in an array.
- Implement the function *isPriorityProvince*: Each time the drone moves to a new province, check if it's a priority location. At the end of the journey, report how many of the priority provinces were visited.
- Implement the function *enqueuePriority* in Queue class: When a neighboring province is a priority province, add it to the queue in a way that it will be dequeued and explored before other non-priority neighbors.

Weather-Restricted Provinces: Due to unfavorable weather conditions, some provinces are inaccessible and must be avoided during exploration.

- Goal: Ensure that the drone does not enter any weather-restricted provinces.
- Implement the function *loadWeatherRestrictedProvinces*: Load the weather-restricted provinces from the given input file and store them in an array.
- Implement the function *isWeatherRestricted*: Before enqueuing a province for exploration, check if it's in the restricted list. If it is, skip that province.

2.3 Step 3: Marking Visited Provinces

- Handling Visited Provinces: You should maintain a visited array that marks provinces as visited after Drone ALPHA moves to them. This array ensures that it explores new routes rather than doubling back to previously visited locations.
- Implement the function *markAsVisited*: This function marks a province as visited. Define a boolean array *visited* where each entry represents whether a province has been visited.
- No Revisits: Once a province has been visited, it cannot be revisited to prevent loops and redundant paths.
- Implement the function *isVisited*: This function checks if a province has already been visited.
- Resetting the Visited Array: This function will ensure that all provinces are marked as unvisited at the beginning of each simulation or exploration phase.

3 Route Exploration

Mission Context

Drone ALPHA must navigate across Türkiye's provinces, exploring every reachable path within its fuel limit. The Stack allows it to backtrack when it reaches a dead-end, while the Queue helps it manage neighboring provinces efficiently. You will now use both structures to simulate a dynamic journey that respects travel constraints and maximizes provincial coverage.

The exploration process involves a few key steps: using the Stack to backtrack, enqueuing neighboring provinces, managing constraints, and tracking the visited status of each province.
Stack with Distance Constraints: Each time Drone ALPHA moves to a new province within the distance limit, it pushes the province to the stack, tracking the route history. If no further moves are possible within the range, the stack allows Drone ALPHA to backtrack.
Queue with Neighboring Provinces: Drone ALPHA enqueues all unvisited neighboring provinces within range, ensuring it evaluates each reachable path. As provinces are dequeued, they are assessed for potential exploration within the given constraints.

Implementation Tips

- Begin exploration from the starting province (Ankara). Track visited provinces to prevent revisits.
- For each province in the journey, enqueue neighboring provinces within the distance limit that meet all conditions.
- Priority provinces should be favored in the exploration, while restricted provinces must be avoided. Implement checks to prioritize or exclude provinces based on these constraints.

4 Output, Results, and Stopping Condition

Mission Context

As Drone ALPHA explores Türkiye's provinces, its journey must eventually come to a logical conclusion. The stopping condition will ensure that Drone ALPHA doesn't explore indefinitely and that it reaches a point where no further moves are available. The final output should display the route taken, the number of provinces visited, and the total distance covered.

Stopping Condition: The exploration must stop when Drone ALPHA has exhausted all possible routes. This condition will be met when:

- Stack and Queue are Empty: The drone has no further provinces to explore, and backtracking has been completed. This signals the end of the exploration, as all possible paths have been tried.
- No Unvisited Neighbors within Range: For any province, if no new provinces are reachable within the specified distance, and the drone cannot backtrack any further, the exploration should end.

Final Output and Results: Once the stopping condition is met, the results should display:

- Every time skipping a weather-restricted province.
- Total Number of Provinces Visited.
- Sequence of Provinces in the Route (in the order they were visited).
- Total Distance Covered (sum of distances between visited provinces).
- Priority Province Summary: Display the total number of priority provinces visited. List each priority province along with its visit status (e.g., "Visited" or "Not Visited"). Provide a completion message if all priority provinces were visited or a warning message if some were missed.
- You'll define helper functions within RoutePlanner to calculate and display the journey summary.

Below is an example of what the program output should look like:

```
Province Amasya is weather-restricted. Skipping.
Province Artvin is weather-restricted. Skipping.
Province Artvin is weather-restricted. Skipping.
Province Rize is weather-restricted. Skipping.
Province Artvin is weather-restricted. Skipping.
Province Amasya is weather-restricted. Skipping.
----------------------------
Journey Completed!
----------------------------
```

```
Total Number of Provinces Visited: 52
Total Distance Covered: 1450 km
Route Taken:
Ankara -> Konya -> Antalya -> Isparta -> Burdur -> ...

Priority Provinces Status:
- Ankara (Visited)
- Izmir (Not Visited)
- Istanbul (Visited)
- Bursa (Visited)
- Trabzon (Not Visited)

Total Priority Provinces Visited: 3 out of 5
Warning: Not all priority provinces were visited.
```

If all priority provinces were visited, display:

```
Total Priority Provinces Visited: 5 out of 5
Success: All priority provinces were visited.
```

5 Assignment Implementation Tasks and Requirements

This section outlines the classes and functions students must implement. You should use the starter code structure provided and adhere to naming conventions and function signatures to ensure consistency and avoid compilation issues. You may add auxiliary functions if needed.

5.1 Class Descriptions and Requirements

Below is an explanation of each method provided in the starter code, which you are required to implement:

5.1.1 Stack Class

A static array-based stack implementation to manage backtracking for Drone ALPHA.

Constructor: Initialize top to indicate an empty stack.

Functions:

- push(): Add a province to the top of the stack. Handle overflow by printing an error message if needed.
- pop(): Remove and return the top element, or indicate if the stack is empty.
- peek(): Return the top element without removing it.
- isEmpty(): Return true if the stack is empty.
- getSize(): Return the number of elements in the stack.

5.1.2 Queue Class

A static array-based, circular queue implementation to handle the list of provinces Drone ALPHA will explore.

Constructor: Initialize front and rear indices to to indicate an empty queue.

Functions:

- enqueue(): Add a province to the rear of the queue. Check for overflow and handle it, take advantage of the circular behavior.
- dequeue(): Remove and return the front element, or indicate if the queue is empty.
- peek(): Return the front element without removing it.
- isEmpty(): Return true if the queue is empty; otherwise, false.
- enqueuePriority(): Add a priority neighboring province to the queue in a way that it will be dequeued and explored before other non-priority neighbors.

5.1.3 Map Class

Represents the Turkish provinces map, including distances between each province and the ability to check for constraints.

Constructor: Initialize the distance matrix and mark all provinces as unvisited.

Functions:

- loadDistanceData(): Load distance data from a CSV file into distanceMatrix. Each cell contains the distance between two provinces. Handle cases where cells indicate no direct connection.
- isWithinRange(): Return true if the distance between two provinces is within maxDistance; otherwise, false.
- markAsVisited(): Mark a specific province as visited.
- isVisited() const: Check if a province has already been visited.
- resetVisited(): Mark all provinces as unvisited.
- getDistance(): Return the distance between two provinces or indicate if no direct route exists.

5.1.4 RoutePlanner Class

Controls the main exploration logic, integrating the stack and queue structures to manage Drone ALPHA's journey.

Constructor: Load the map, reset visited status, and initialize parameters for exploration.

Functions:

- loadPriorityProvinces(): Load priority provinces from "priority_provinces.txt" file containing the priority city names and indices.
- loadWeatherRestrictedProvinces(): Load weather-restricted provinces from "weather_restricted_provinces.txt" file containing the restricted city names and indices.
- isPriorityProvince(): Check if a province is a priority province.
- isWeatherRestricted(): Check if a province is weather-restricted.
- exploreRoute(): This will be the main function to begin the journey from starting city. Mark it as visited, push it onto the stack, and call exploreFromProvince() to start the exploration.
- exploreFromProvince(): Enqueue unvisited neighboring provinces and check distance constraints before moving.
- enqueueNeighbors(): Add reachable, unvisited priority and non-priority neighboring provinces to the queue.
- backtrack(): Move back to previous provinces if there are no new moves left.
- isExplorationComplete(): Check the stopping conditions.
- displayResults(): Print the total distance covered, number of provinces visited, and the sequence of provinces (by name), visit status of the priority provinces (by name) in the journey.

5.2 Must-Use Starter Codes

You MUST use the given starter (template) code. You should prepare a ZIP file named b<studentID>.zip containing the all headers and classes.

Compiling and Running the Assignment

Your implementation will be compiled and tested with the following commands. Please ensure your code is compatible with these steps:

```
$ g++ -std=c++11 *.cpp *.h -o DroneALPHA
```

After successful compilation, you can run the program with the following command:

```
$ ./DroneALPHA max_distance starting_city_index distance_file
priority_provinces_file weather_restricted_provinces_file
```

Here's an example of running the program with specific inputs:

```
$ ./DroneALPHA 250 5 distance_data.csv priority_provinces.txt
weather_restricted_provinces.txt
```

Ensure your program reads the command-line arguments in the specified order: maximum distance, starting city index, and file paths.

5.3 Grading Policy

Implementation of Exploration Logic: 80%

- Proper initialization and setup of the map, queue, and stack structures. 20%
- Accurate handling of the given travel constraints. 20%
- Exploration and backtracking implementation. 25%
- Effective usage of queue and stack for exploration. 15%

Memory Management: 5%

- No memory leaks or errors.

Output Testing: 15%

- Correct output formatting based on the given case, showing complete exploration.

5.4 Important Submission Information

- Deadline: Submit your assignment by **Sunday, 01.12.2024 (23:59:59)**.
- Save all your work until the assignment is fully graded.
- The solution you submit must be your original, individual work. Duplicate or similar assignments will be considered as cheating.
- You can ask questions via Piazza Page. Please ensure you are aware of everything discussed there.
- Test your code using Tur3BoGrader before submission. Testing on Tur3BoGrader does not count as submission.
- Submit it to submit.cs.hacettepe.edu.tr, where you will be assigned a submission.

Submission Format

You MUST use the provided starter code template. All classes should be placed directly in your .zip archive. Submit only in .zip format (no .rar or other archive formats are supported). Submit your code as a single .zip archive named as: **b<studentID>.zip**

Inside the .zip archive, include the following files directly (not in any folders):
Map.h <FILE>
Map.cpp <FILE>
Queue.h <FILE>
Queue.cpp <FILE>
Stack.h <FILE>
Stack.cpp <FILE>
RoutePlanner.h <FILE>
RoutePlanner.cpp <FILE>

Academic Integrity

All work on assignments must be done individually unless stated otherwise. You are encouraged to discuss with your classmates about the given assignments, but these discussions should be carried out in an abstract way. That is, discussions related to a particular solution to a specific problem (either in actual code or in the pseudocode) will not be tolerated. In short, turning in someone else's work, in whole or in part, as your own will be considered as a violation of academic integrity. Please note that the former condition also holds for online/AI sources. Make use of them responsibly, refrain from generating the code you are asked to implement. Remember that we also have have access to such tools, making it easier to detect such cases.