A diagram of a college

AI-generated content may be incorrect.

Class Diagram for College Data, Souvenir Data, and Souvenir Purchase

A screen shot of a computer code

AI-generated content may be incorrect.

The function findClosestCollege searches through a vector of CollegeData entries to find the closest unvisited college from a given currentCollege. It maintains a minimum distance tracker and an index of the closest college.

* float minDistance = std::numeric\_limits<float>::max(); → **O(1)**
* nextIndex = -1; → **O(1)**
* for (size\_t i = 0; i < data.size(); ++i) - This runs **O(N)** times, where N is the number of elements in data.
* visitedCampuses.find(data[i].collegeEnd) → **O(1)** (unordered\_set provides average O(1) lookup time)
*  data[i].collegeStart == currentCollege → **O(1)**
*  data[i].distance < minDistance → **O(1)**
*  Updating minDistance and nextIndex → **O(1)**

## Overall Time Complexity

* The dominant term is the **loop over data**, which runs **O(N)** times.
* Inside the loop, all operations execute in **constant time O(1)**.

Thus, the **overall time complexity** of findClosestCollege is: **O(N)**

## Space Complexity

* The function uses only a few additional variables (minDistance, nextIndex), all of which are O(1).
* It takes references to data and visitedCampuses, but it does not create new data structures.

**Thus, the space complexity is: O(1)**

## Data Structures Used

**1. std::vector<CollegeData> data;**

* **Type:** std::vector
* **Purpose:** Stores all possible college connections, including distances.
* **Characteristics:**
  + **Dynamic Array:** std::vector is a **dynamically resizable array**.
  + **Access Time:** Provides **O(1)** access time for indexing elements.
  + **Insertion at End:** push\_back() is **O(1)** on average.
  + **Memory Usage:** Stores up to stops elements, leading to **O(stops) space complexity**.

**2. std::unordered\_set<std::string> visitedCampuses;**

* **Type:** std::unordered\_set
* **Purpose:** Keeps track of visited colleges to prevent revisits.
* **Characteristics:**
  + **Hash Table-Based Set:** Uses a **hash table** for storage.
  + **Insertion & Lookup:** Average **O(1)** for insertions (insert()) and lookups (find()).
  + **Prevents Duplicates:** Ensures a college is not revisited.
  + **Memory Usage:** Stores up to stops elements → **O(stops) space complexity**.

**A screen shot of a computer code

AI-generated content may be incorrect.**

This function constructs an efficient route visiting the closest colleges, ensuring that no college is revisited. It runs for a specified number of stops and calls findClosestCollege at each iteration.

* Initialization (Constant Time)
  + std::vector<CollegeData> route;
  + std::unordered\_set<std::string> visitedCampuses;
  + std::string currentCollege = startCollege;
  + totalDistance = 0.0;
  + These operations all take **O(1)** time.
* Loop Over Stops (Iterative)
  + for (int i = 0; i < stops; ++i)
  + The loop runs **O(stops)** times.
  + Calling findClosestCollege (**O(N)**)
    - float distance = findClosestCollege(currentCollege, data, visitedCampuses, nextIndex);
  + Operations Inside the Loop (Constant Time)
    - if (nextIndex == -1) break; // **O(1)**
    - visitedCampuses.insert(currentCollege); // **O(1)**
    - route.push\_back(data[nextIndex]); // **O(1)**
    - totalDistance += distance; // **O(1)**
    - currentCollege = data[nextIndex].collegeEnd; // **O(1)**

## Overall Time Complexity

* The dominant cost comes from the **loop over stops**, where in each iteration we call findClosestCollege, which runs in **O(N)** time.
* Since the loop executes **O(stops)** times, the total complexity is: **O(stops×N)**

If **stops = N**, the worst-case complexity is: **O(N²)**

Thus, the **worst-case time complexity** is **O(N²)**.

## Space Complexity

* The function maintains:
  + std::vector<CollegeData> route (stores at most stops elements) → **O(stops)**
  + std::unordered\_set<std::string> visitedCampuses (stores visited colleges, at most stops) → **O(stops)**
  + Other variables (currentCollege, totalDistance, nextIndex) are **O(1)**.

Thus, the **overall space complexity** is: **O(stops)**

## Data Structures Used

**1. std::vector<CollegeData> route;**

* **Type:** std::vector
* **Purpose:** Stores the sequence of CollegeData objects representing the chosen route.
* **Characteristics:**
  + **Dynamic Array:** std::vector is a **dynamically resizable array**.
  + **Access Time:** Provides **O(1)** access time for indexing elements.
  + **Insertion at End:** push\_back() is **O(1)** on average.
  + **Memory Usage:** Stores up to stops elements, leading to **O(stops) space complexity**.

**2. std::unordered\_set<std::string> visitedCampuses;**

* **Type:** std::unordered\_set
* **Purpose:** Keeps track of visited colleges to prevent revisits.
* **Characteristics:**
  + **Hash Table-Based Set:** Uses a **hash table** for storage.
  + **Insertion & Lookup:** Average **O(1)** for insertions (insert()) and lookups (find()).
  + **Prevents Duplicates:** Ensures a college is not revisited.
  + **Memory Usage:** Stores up to stops elements → **O(stops) space complexity**.

A screen shot of a computer code

AI-generated content may be incorrect.

The function planEfficientTrip determines an efficient route to visit a given set of colleges while ensuring that each college is visited only once.

* Creating Data Structures: **O(1)**
  + std::vector<CollegeData> route; // O(1)
  + std::unordered\_set<std::string> visitedCampuses; // O(1)
  + totalDistance = 0.0; // O(1)
* Checking if collegesToVisit is empty: **O(1)**
  + if (collegesToVisit.isEmpty()) {
  + return route;}
* Converting QStringList to std::unordered\_set: **O(K)**
  + std::unordered\_set<std::string> remainingColleges;
  + for (int i = 1; i < collegesToVisit.size(); ++i) {
  + remainingColleges.insert(collegesToVisit[i].toStdString()); // O(1) per insertion}
* While Loop Runs Until remainingColleges is Empty: **Worse Case O(K)**
  + while (!remainingColleges.empty()) {
* Finding Closest College (findClosestCollege2()): **O(N), each iteration O(K x N)**
  + float distance = findClosestCollege2(currentCollege, data, visitedCampuses, nextIndex, remainingColleges);
* Handling Cases When No Valid College is Found: **Worst Case O(K² × N) (if every visited college is checked)**
* Other Operations in the Loop: **O(1)**
  + route.push\_back(data[nextIndex]); // O(1)
  + totalDistance += distance; // O(1)
  + visitedCampuses.insert(currentCollege); // O(1)
  + remainingColleges.erase(currentCollege); // O(1)

## Overall Time Complexity

* **Best Case:** **O(K × N)** (if findClosestCollege2() always finds a valid next college immediately).
* **Worst Case:** **O(K² × N)** (when no valid college is found, and extra searches are needed).

## Space Complexity

* **Total Space Complexity: O(K + N)** (dominated by storage for route, visitedCampuses, remainingColleges, and data).

## Data Structures Used

**std::vector<CollegeData> route;**

* **Type:** std::vector
* **Purpose:** Stores the ordered list of college connections forming the planned trip.
* **Operations Used:**
  + push\_back(data[nextIndex]); → **O(1)** (appending to the vector)
  + return route; → **O(1)** (returning the vector)
* **Characteristics:**
  + **Dynamic Array:** std::vector is a dynamically resizable array.
  + **Efficient Access:** Provides **O(1)** access time for indexing elements.
  + **Efficient Insertion at End:** push\_back() is **O(1)** on average.
* **Space Complexity:** **O(K)** (where K is the number of stops in the trip)

**2. std::unordered\_set<std::string> visitedCampuses;**

* **Type:** std::unordered\_set
* **Purpose:** Tracks colleges that have already been visited to prevent revisits.
* **Operations Used:**
  + visitedCampuses.insert(currentCollege); → **O(1)**
  + visitedCampuses.count(visited) > 0; → **O(1)** (membership check)
* **Characteristics:**
  + **Hash Table-Based Set:** Uses a **hash table** for fast lookups.
  + **Efficient Insertions & Lookups:**
    - Insertion (insert()) and lookup (find()) take **O(1)** on average.
    - Worst-case **O(N)** (if hash collisions occur, though unlikely).
* **Space Complexity:** **O(K)** (since it stores at most K visited colleges).

**3. std::unordered\_set<std::string> remainingColleges;**

* **Type:** std::unordered\_set
* **Purpose:** Stores the set of **colleges yet to be visited**.
* **Operations Used:**
  + remainingColleges.insert(collegesToVisit[i].toStdString()); → **O(1)**
  + remainingColleges.erase(currentCollege); → **O(1)** (removing a visited college)
* **Characteristics:**
  + **Fast lookups and deletions** due to hash table implementation.
  + Allows **quick membership tests** before inserting elements into the min-heap.
* **Space Complexity:** **O(K)** (since it stores at most K unvisited colleges).

**4. std::vector<CollegeData> data;**

* **Type:** std::vector
* **Purpose:** Stores all **possible college connections** (i.e., available paths between colleges).
* **Operations Used:**
  + **Linear search to find closest colleges** (findClosestCollege2())
* **Characteristics:**
  + **Dynamic Array:** Stores **N** CollegeData elements.
  + **Search Complexity:** Each iteration performs **O(N)** operations.
* **Space Complexity:** **O(N)** (where N is the number of available college connections).