NAME: ADITYA JETHANI
DIV-3 G-5
ROLL NO: 21BCP166
DAA Assignment 4

AIM: Making a city database with unordered lists: Array based and linked with basic functionalities like inserting, searching, deleting and printing the database.

CODE for array-based list:

```
#include <iostream>
#include <string>
#include <vector>
#include <cmath>

struct City {
    std::string name;
    int x;
    int y;
};
std::vector<City> cities;
void insertCity(const std::string& name, int x, int y) {
    City city;
    city.name = name;
    city.x = x;
    city.y = y;
    cities.push_back(city);
}
void deleteCity(const std::string& name) {
    for (int i = 0; i < cities.size(); i++) {
        if (cities[i].name == name) {
            cities.erase(cities.begin() + i);
```

```
            i--;
        }
    }
}
void deleteCity(int x, int y) {
    for (int i = 0; i < cities.size(); i++) {
        if (cities[i].x == x && cities[i].y == y) {
            cities.erase(cities.begin() + i);
            i--;
        }
    }
}
std::vector<City> searchCity(const std::string& name) {
    std::vector<City> results;
    for (int i = 0; i < cities.size(); i++) {
        if (cities[i].name == name) {
            results.push_back(cities[i]);
        }
    }
    return results;
}
std::vector<City> searchCity(int x, int y) {
    std::vector<City> results;
    for (int i = 0; i < cities.size(); i++) {
        if (cities[i].x == x && cities[i].y == y) {
            results.push_back(cities[i]);
        }
    }
    return results;
}
void printCitiesWithinDistance(int x, int y, int distance) {
    for (int i = 0; i < cities.size(); i++) {
        int dx = cities[i].x - x;
        int dy = cities[i].y - y;
        if (std::sqrt(dx * dx + dy * dy) <= distance) {
            std::cout << cities[i].name << " " << cities[i].x << " " <<
cities[i].y << std::endl;
        }
    }
}
```

```
int main() {
    int choice;
    std::string name;
    int x, y, distance;
    std::vector<City> results;
    while (true) {
        std::cout << "1. Insert a city" << std::endl;
        std::cout << "2. Delete a city" << std::endl;
        std::cout << "3. Search for a city" << std::endl;
        std::cout << "4. Print cities within a distance" << std::endl;
        std::cout << "5. Exit" << std::endl;
        std::cout << "Enter your choice: ";
        std::cin >> choice;
        if (choice == 1) {
            std::cout << "Enter the name of the city: ";
            std::cin >> name;
            std::cout << "Enter the x coordinate of the city: ";
            std::cin >> x;
            std::cout << "Enter the y coordinate of the city: ";
            std::cin >> y;
            insertCity(name, x, y);
        } else if (choice == 2) {
            std::cout << "Enter the name of the city: ";
            std::cin >> name;
            deleteCity(name);
        } else if (choice == 3) {
            std::cout << "Enter the name of the city: ";
            std::cin >> name;
            results = searchCity(name);
            for (int i = 0; i < results.size(); i++) {
                std::cout << results[i].name << " " << results[i].x << " "
<< results[i].y << std::endl;
            }
    } //print cities within a distance of a given city and if theres no
city with that name print no city found
        else if (choice == 4) {
            std::cout << "Enter the name of the city: ";
            std::cin >> name;
            std::cout << "Enter the distance: ";
            std::cin >> distance;
```

```
            results = searchCity(name);
            if (results.size() == 0) {
                std::cout << "No city found" << std::endl;
            } else {
                for (int i = 0; i < results.size(); i++) {
                    printCitiesWithinDistance(results[i].x, results[i].y,
distance);
                }
            }

        } else if (choice == 5) {
            break;
        }
        else{
            std::cout<<"Invalid choice"<<std::endl;
        }
    }
    return 0;
}
```

OUTPUT:

```
Enter your choice: 1
Enter the name of the city: ahmedabas
Enter the x coordinate of the city: 14
Enter the y coordinate of the city: 13
1. Insert a city
2. Delete a city
3. Search for a city
4. Print cities within a distance
5. Exit
Enter your choice: 4
Enter the name of the city: surat
Enter the distance: 100
surat 11 12
ahmedabas 14 13
1. Insert a city
2. Delete a city
3. Search for a city
4. Print cities within a distance
5. Exit
```

(The selected part is the main implementation)

CODE FOR LINKED LIST:

```
#include <iostream>
#include <unordered_map>
using namespace std;
struct City {
    int x;
    int y;
};
class CityDatabase {
    public:
        unordered_map<string, City> cities;

        void insert_city(string name, int x, int y) {
            City city = {x, y};
            cities[name] = city;
        }

        void delete_city_by_name(string name) {
            cities.erase(name);
        }

        City* search_city_by_name(string name) {
            if (cities.find(name) != cities.end())
                return &cities[name];
            else
                return nullptr;
        }

        void print_database() {
            for (auto& city : cities) {
                cout << city.first << ": (" << city.second.x << ", " <<
city.second.y << ")" << endl;
            }
        }
        unordered_map<string, City> search_city_by_distance(int x, int y,
int distance) {

            unordered_map<string, City> result;
            for (auto& city : cities) {
```

```
                int dx = abs(city.second.x - x);
                int dy = abs(city.second.y - y);
                if(dx * dx + dy * dy <= distance * distance) {
                    result[city.first] = city.second;
                }
            }
            return result;
        }
};
int main() {
    CityDatabase db;
    int choice;
    //display menu
    cout << "1. Insert city" << endl;
    cout << "2. Delete city" << endl;
    cout << "3. Search city by name" << endl;
    cout << "4. Search city by distance" << endl;
    cout << "5. Print database" << endl;
    cout << "6. Exit" << endl;
    cout << "Enter your choice: ";
    cin >> choice;

    while (choice != 6) {
        switch (choice) {
            case 1: {
                string name;
                int x, y;
                cout<<"Enter city name:"<<endl;
                cin>>name;
                cout<<"Enter x coordinate:"<<endl;
                cin>>x;
                cout<<"Enter y coordinate:"<<endl;
                cin>>y;

                db.insert_city(name, x, y);
                break;
            }
            case 2: {
                string name;
                cout << "Enter name: ";
```

```
                cin >> name;
                db.delete_city_by_name(name);
                break;
            }
            case 3: {
                string name;
                cout << "Enter name: ";
                cin >> name;
                City* city = db.search_city_by_name(name);
                if (city != nullptr) {
                    cout << "Found " << name << ": (" << city->x << ", "
<< city->y << ")" << endl;
                } else {
                    cout << "Not found" << endl;
                }
                break;
            }
            case 4: {
                int x, y, distance;
                cout << "Enter x, y, distance: ";
                cin >> x >> y >> distance;
                unordered_map<string, City> result =
db.search_city_by_distance(x, y, distance);
                for (auto& city : result) {
                    cout << city.first << ": (" << city.second.x << ", "
<< city.second.y << ")" << endl;
                }
                break;
            }
            case 5: {
                db.print_database();
                break;
            }
            default: {
                cout << "Invalid choice" << endl;
            }
        }
        cout << "Enter your choice: ";
        cin >> choice;
    }}
```

OUTPUT:

```
PS C:\Users\jetha> cd "c:\Aditya\College Code\DAA lab\P4\" ; i
1. Insert city
2. Delete city
3. Search city by name
4. Search city by distance
5. Print database
6. Exit
Enter your choice: 1
Enter city name:
surat
Enter x coordinate:
10
Enter y coordinate:
20
Enter your choice: 3
Enter name: surat
Found surat: (10, 20)
Enter your choice: 6
```

1.1) ANALYSIS (linked list) :

1) The running time of each operation in the code depends on the number of cities in the database and the operations being performed.

Insert city: O(1), as unordered_map insert operation takes constant time on average.
Delete city by name: O(1), as unordered_map erase operation takes constant time on average.
Search city by name: O(1), as unordered_map find operation takes constant time on average.
Search city by distance: O(n), as the function needs to loop through all cities in the database and calculate the distance for each city.

Print database: O(n), as the function needs to loop through all cities in the database to print them.
The overall time complexity of the code will depend on the number of cities in the database and the operations being performed.

1.2) ANALYSIS (array-based):

The time complexity of the program depends on the size of the cities vector, which stores all the cities that have been added to the system. The following is the time complexity of each function in the program:

insertCity: O(1) - This function simply appends a city to the end of the cities vector, so the time complexity is constant.

deleteCity: O(n) - This function has two overloads. In the first overload, which takes the name of the city, it loops through the entire cities vector to find the cities with the given name, and deletes them one by one. In the second overload, which takes the x and y coordinates, it also loops through the entire cities vector to find the city with the given x and y coordinates, and deletes it. In both cases, the time complexity is proportional to the size of the cities vector.

searchCity: O(n) - This function has two overloads, both of which loop through the entire cities vector to find the cities with the given name or x and y coordinates. The time complexity is proportional to the size of the cities vector.

printCitiesWithinDistance: O(n) - This function loops through the entire cities vector to find the cities within a certain distance from a given city, and prints them. The time complexity is proportional to the size of the cities vector.

main: O(n^2) - This is the main function of the program, which calls all the above functions. The time complexity is proportional to the square of the size of the cities vector, because there are nested loops in the function.

2.1) ADVANTAGES AND DISADVANTAGES OF VECTOR:

Advantages:

- Convenience: shorter workouts are more manageable and can be fit into a busy schedule.
- Time-saving: shorter workouts take less time, making them a good option for those with limited time.
- Intensity: short, high-intensity workouts can be more effective for some people than longer, low-intensity workouts.

Disadvantages:

- Reduced calorie burn: shorter workouts may burn fewer calories compared to longer workouts.
- Limited progress: shorter workouts may limit the progress that can be made in terms of strength, endurance, and fitness.
- Need for frequency: shorter workouts may need to be done more frequently to achieve the same results as a longer workout.

2.2) ADVANTAGES AND DISADVANTAGES OF LINKED LIST:

Advantages:

- Easy to add and remove cities using insert_city and delete_city_by_name functions.

- Fast city search using unordered_map container.
- search_city_by_distance function returns cities within the specified distance.
- Clean and simple code with clear function names.

Disadvantages:

- Only stores 2D coordinates for a city and doesn't store any other information.
- Distance calculation in search_city_by_distance function is not accurate as it uses the Manhattan distance instead of actual distance.
- No error handling for invalid inputs, for example, negative coordinates or invalid menu choice.

3.1) IF WE STORE THE RECORDS ALPHABETICALLY (Linked List):

No, storing the records in alphabetical order by city name would not speed up any of the operations. An unordered_map is an associative container that stores elements in no particular order, but provides constant time average complexity for search, insertion and deletion operations, regardless of the order of the elements. If you need to search for elements based on a specific order, you can use an ordered data structure like std::map instead.

3.2) IF WE STORE THE RECORDS ALPHABETICALLY (Array based list):

No, storing records in alphabetical order by city name will not necessarily speed up any of the operations. The time complexity of operations such as search, insertion, and deletion depends on the underlying data structure and algorithms used, rather than the order of elements in the list. For example, searching for a record in an unsorted list will have a time

complexity of O(n), while a sorted list can be searched in O(log n) time using binary search.

4.1) WOULD KEEPING THE LIST IN ALPHABETICAL ORDER SLOW DOWN ANY OPERATION? (Linked List)

No, the unordered_map data structure used in the CityDatabase class does not maintain any order of the elements stored in it. The elements are stored in a hash table and are accessed using the key value. Keeping the list in alphabetical order would not affect the insert, delete, or search operations, but it would slow down the print operation as the elements would need to be sorted before printing.

4.2) WOULD KEEPING THE LIST IN ALPHABETICAL ORDER SLOW DOWN ANY OPERATION? (Array Based list)

Yes, keeping the list in alphabetical order would slow down some of the operations. For example, inserting a new city in the list would require shifting all the elements after the insertion point to make space for the new city, which could be time-consuming if the list is long. Similarly, searching for a city by name would require a linear search instead of a constant time lookup, which would make the search operation slower. On the other hand, deleting a city by name would be faster if the list is sorted, since the elements after the deletion point would not need to be shifted.