## Meet Jain, 23BCP093

**DAA Lab 4: City Database**

**Theory:**

Array-based vs Linked List-based Data Structures

Array-based:

Advantages: Fast access (O(1)) via index, memory efficient (contiguous storage).

Disadvantages: Fixed size, slow insertions and deletions (O(n)), resizing is costly.

Linked List-based:

Advantages: Dynamic size, efficient insertions and deletions (O(1)) at head/tail.

Disadvantages: Slow access (O(n)), higher memory usage due to pointers.

**Code:**

**Array:**

#include <bits/stdc++.h>

using namespace std;

#define MAX\_CITIES 100

struct City {

    string name;

    double x, y;

};

City cities[MAX\_CITIES];

int size = 0;

void insertCity() {

    if (size >= MAX\_CITIES) {

        cout << "Database is full. Cannot insert more cities.\n";

        return;

    }

    string name;

    double x, y;

    cout << "Enter city name: ";

    cin >> name;

    cout << "Enter x and y coordinates: ";

    cin >> x >> y;

    cities[size++] = {name, x, y};

    cout << "City added successfully.\n";

}

void deleteByName() {

    string name;

    cout << "Enter city name to delete: ";

    cin >> name;

    for (int i = 0; i < size; i++) {

        if (cities[i].name == name) {

            cities[i] = cities[size - 1];

            size--;

            cout << "City deleted successfully.\n";

            return;

        }

    }

    cout << "No city found with name: " << name << endl;

}

void deleteByCoordinates() {

    double x, y;

    cout << "Enter x and y coordinates to delete: ";

    cin >> x >> y;

    for (int i = 0; i < size; i++) {

        if (cities[i].x == x && cities[i].y == y) {

            cities[i] = cities[size - 1];

            size--;

            cout << "City deleted successfully.\n";

            return;

        }

    }

    cout << "No city found at coordinates: (" << x << ", " << y << ")\n";

}

void searchByName() {

    string name;

    cout << "Enter city name to search: ";

    cin >> name;

    for (int i = 0; i < size; i++) {

        if (cities[i].name == name) {

            cout << "City found: " << cities[i].name << " (" << cities[i].x << ", " << cities[i].y << ")\n";

            return;

        }

    }

    cout << "No city found with name: " << name << endl;

}

void searchByCoordinates() {

    double x, y;

    cout << "Enter x and y coordinates to search: ";

    cin >> x >> y;

    for (int i = 0; i < size; i++) {

        if (cities[i].x == x && cities[i].y == y) {

            cout << "City found: " << cities[i].name << " (" << cities[i].x << ", " << cities[i].y << ")\n";

            return;

        }

    }

    cout << "No city found at coordinates: (" << x << ", " << y << ")\n";

}

void printWithinDistance() {

    double x, y, d;

    cout << "Enter x and y coordinates and distance: ";

    cin >> x >> y >> d;

    cout << "Cities within distance " << d << " from (" << x << ", " << y << "):\n";

    for (int i = 0; i < size; i++) {

        double dist = sqrt(pow(cities[i].x - x, 2) + pow(cities[i].y - y, 2));

        if (dist <= d) {

            cout << cities[i].name << " (" << cities[i].x << ", " << cities[i].y << ")\n";

        }

    }

}

int main() {

    int choice;

    while (true) {

        cout << "\nMenu:\n";

        cout << "1. Insert city\n2. Delete by name\n3. Delete by coordinates\n";

        cout << "4. Search by name\n5. Search by coordinates\n6. Print cities within distance\n";

        cout << "7. Exit\nEnter your choice: ";

        cin >> choice;

        switch (choice) {

            case 1: insertCity(); break;

            case 2: deleteByName(); break;

            case 3: deleteByCoordinates(); break;

            case 4: searchByName(); break;

            case 5: searchByCoordinates(); break;

            case 6: printWithinDistance(); break;

            case 7: cout << "Exiting program.\n"; return 0;

            default: cout << "Invalid choice. Try again.\n";

        }

    }

}

**LinkedList:**

#include <bits/stdc++.h>

using namespace std;

struct Node {

    string name;

    double x, y;

    Node\* next;

    Node(string name, double x, double y, Node\* next = nullptr) {

        this->name = name;

        this->x = x;

        this->y = y;

        this->next = next;

    }

};

Node\* createNode(string name, double x, double y) {

    return new Node(name, x, y);

}

void inserthead(Node\*& head, string name, double x, double y) {

    Node\* temp = createNode(name, x, y);

    temp->next = head;

    head = temp;

}

void deleterec(Node\*& head, string name) {

    Node\* ptr = head;

    Node\* prev = nullptr;

    while (ptr) {

        if (ptr->name == name) {

            if (ptr == head) {

                head = head->next;

                delete ptr;

            } else {

                prev->next = ptr->next;

                delete ptr;

            }

            cout << "Record deleted successfully.\n";

            return;

        }

        prev = ptr;

        ptr = ptr->next;

    }

    cout << "No record found with name: " << name << endl;

}

void deleteco(Node\*& head, double x, double y) {

    Node\* ptr = head;

    Node\* prev = nullptr;

    while (ptr) {

        if (ptr->x == x && ptr->y == y) {

            if (ptr == head) {

                head = head->next;

                delete ptr;

            } else {

                prev->next = ptr->next;

                delete ptr;

            }

            cout << "Record deleted successfully.\n";

            return;

        }

        prev = ptr;

        ptr = ptr->next;

    }

    cout << "No record found at coordinates: (" << x << ", " << y << ")" << endl;

}

void printnode(Node\* temp) {

    if (temp)

        cout << temp->name << " (" << temp->x << ", " << temp->y << ")\n";

}

void searchname(Node\* head, string name) {

    Node\* ptr = head;

    while (ptr) {

        if (ptr->name == name) {

            cout << "Record found: ";

            printnode(ptr);

            return;

        }

        ptr = ptr->next;

    }

    cout << "No record found with name: " << name << endl;

}

void searchco(Node\* head, double x, double y) {

    Node\* ptr = head;

    while (ptr) {

        if (ptr->x == x && ptr->y == y) {

            cout << "Record found: ";

            printnode(ptr);

            return;

        }

        ptr = ptr->next;

    }

    cout << "No record found at coordinates: (" << x << ", " << y << ")" << endl;

}

void printrec(Node\* head, double x, double y, double d) {

    Node\* ptr = head;

    bool found = false;

    cout << "Records within radius " << d << ":\n";

    while (ptr) {

        double dist = sqrt(pow(x - ptr->x, 2) + pow(y - ptr->y, 2));

        if (dist <= d) {

            printnode(ptr);

            found = true;

        }

        ptr = ptr->next;

    }

    if (!found) cout << "No records found within the given radius.\n";

}

void displayAll(Node\* head) {

    if (!head) {

        cout << "Database is empty.\n";

        return;

    }

    cout << "City Records:\n";

    while (head) {

        printnode(head);

        head = head->next;

    }

}

int main() {

    Node\* cityList = nullptr;

    int choice;

    while (true) {

        cout << "\n1) Insert Record\n2) Delete by Name\n3) Delete by Coordinates\n";

        cout << "4) Search by Name\n5) Search by Coordinates\n6) Display All\n";

        cout << "7) Find Cities in Range\n8) Exit\nEnter choice: ";

        cin >> choice;

        if (choice == 8) {

            cout << "Exiting...\n";

            break;

        }

        string name;

        double x, y, d;

        switch (choice) {

            case 1:

                cout << "Enter City Name: ";

                cin >> name;

                cout << "Enter Coordinates (x y): ";

                cin >> x >> y;

                inserthead(cityList, name, x, y);

                break;

            case 2:

                cout << "Enter City Name: ";

                cin >> name;

                deleterec(cityList, name);

                break;

            case 3:

                cout << "Enter Coordinates (x y): ";

                cin >> x >> y;

                deleteco(cityList, x, y);

                break;

            case 4:

                cout << "Enter City Name: ";

                cin >> name;

                searchname(cityList, name);

                break;

            case 5:

                cout << "Enter Coordinates (x y): ";

                cin >> x >> y;

                searchco(cityList, x, y);

                break;

            case 6:

                displayAll(cityList);

                break;

            case 7:

                cout << "Enter Reference Point (x y): ";

                cin >> x >> y;

                cout << "Enter Search Radius: ";

                cin >> d;

                printrec(cityList, x, y, d);

                break;

            default:

                cout << "Invalid choice. Try again.\n";

        }

    }

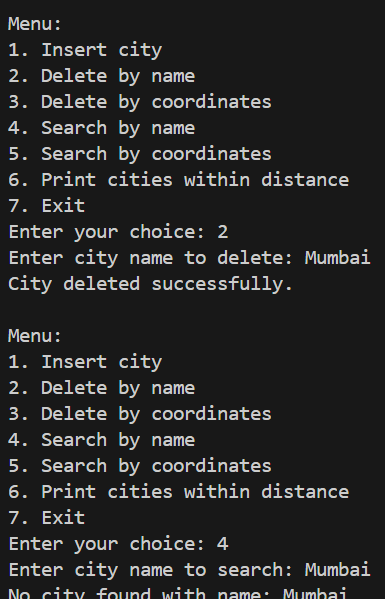
    return 0;

}

**Output:**

**A computer screen shot of a black screen

AI-generated content may be incorrect.**

****

**Comparison of Time Complexities:**

* Insertions:
  + Array-based: O(1) for inserting at the end.
  + Linked List-based: O(1) for inserting at the head.

Both implementations are efficient for insertions, but the array-based implementation would require O(n) if it needed to resize the array (which isn't explicitly handled in the code).

* Deletions:
  + Array-based: O(n), due to the need to search and then shift elements.
  + Linked List-based: O(n), because it requires a traversal to find the node to delete.

In both cases, the deletion operation requires linear time due to the need to search for the element.

* Search Operations:
  + Array-based: O(n), because it performs a linear search.
  + Linked List-based: O(n), because it also performs a linear search.
* Display Operations:
  + Array-based: O(n), because it has to print each city.
  + Linked List-based: O(n), because it also prints each city in the linked list.
* Finding Cities Within Range:
  + Array-based: O(n), because it needs to check each city's distance.
  + Linked List-based: O(n), because it also checks the distance for each city.

Key Observations:

* The time complexity for the major operations (insert, delete, search, print, etc.) is O(n) for both array-based and linked list-based implementations, due to the need to traverse the list/array.
* Insertion at the head (linked list) or end of the array (array-based) is an efficient O(1) operation.
* Memory Usage:
  + The array-based approach can become inefficient if resizing is required, especially when the array reaches its maximum size.
  + The linked list approach is more flexible in terms of memory management as it dynamically allocates and deallocates memory for nodes. However, it uses more memory due to the need to store pointers in each node.

In conclusion, both approaches have similar time complexity for most operations, but the linked list provides more flexibility in terms of dynamic memory usage, whereas the array implementation may have performance issues if the array size exceeds its limit.