**Practical 4: City Databases using Linked List**

**Aim**

Implement a **city database** using **unordered lists**. Each database record contains the name of the city (a string of arbitrary length) and the coordinates of the city expressed as integer x and y coordinates. Your program should allow following functionalities:

1. Insert a record
2. Delete a record by name or coordinate
3. Search a record by name or coordinate
4. Point all records within a given distance of a specified point.

Implement the database using an **array-based list** implementation, and then a **linked list** implementation

**Program**

1. **Array based Implementation**

public class CityDatabaseArray {  
  
 private static final int *INITIAL\_CAPACITY* = 10;  
 private String[] cityNames;  
 private int[] xCoords;  
 private int[] yCoords;  
 private int size;  
  
 public CityDatabaseArray() {  
 cityNames = new String[*INITIAL\_CAPACITY*];  
 xCoords = new int[*INITIAL\_CAPACITY*];  
 yCoords = new int[*INITIAL\_CAPACITY*];  
 size = 0;  
 }  
  
 *// Inserts a record with the given name and coordinates* public void insert(String name, int x, int y) {  
 if (size >= cityNames.length) {  
 resize();  
 }  
 cityNames[size] = name;  
 xCoords[size] = x;  
 yCoords[size] = y;  
 size++;  
 }  
  
 *// Deletes a record with the given name or coordinates* public void delete(String nameOrCoord) {  
 for (int i = 0; i < size; i++) {  
 if (cityNames[i].equals(nameOrCoord) || (xCoords[i] + "," + yCoords[i]).equals(nameOrCoord)) {  
 cityNames[i] = cityNames[size - 1];  
 xCoords[i] = xCoords[size - 1];  
 yCoords[i] = yCoords[size - 1];  
 size--;  
 return;  
 }  
 }  
 }  
  
 *// Searches for a record with the given name or coordinates and returns its index, or -1 if not found* public int search(String nameOrCoord) {  
 for (int i = 0; i < size; i++) {  
 if (cityNames[i].equals(nameOrCoord) || (xCoords[i] + "," + yCoords[i]).equals(nameOrCoord)) {  
 return i;  
 }  
 }  
 System.*out*.println("City is:");  
 return -1;  
 }  
  
 *// Prints all records within the given distance of the specified point* public void printNearby(int x, int y, double distance) {  
 for (int i = 0; i < size; i++) {  
 double dx = xCoords[i] - x;  
 double dy = yCoords[i] - y;  
 double dist = Math.*sqrt*(dx\*dx + dy\*dy);  
 if (dist <= distance) {  
 System.*out*.println(cityNames[i] + " (" + xCoords[i] + "," + yCoords[i] + ")");  
 }  
 }  
 }  
  
 *// Resizes the arrays to twice their current capacity* private void resize() {  
 int newCapacity = 2 \* cityNames.length;  
 String[] newCityNames = new String[newCapacity];  
 int[] newXCoords = new int[newCapacity];  
 int[] newYCoords = new int[newCapacity];  
 for (int i = 0; i < size; i++) {  
 newCityNames[i] = cityNames[i];  
 newXCoords[i] = xCoords[i];  
 newYCoords[i] = yCoords[i];  
 }  
 cityNames = newCityNames;  
 xCoords = newXCoords;  
 yCoords = newYCoords;  
 }

public static void main(String[] args) {  
 CityDatabaseArray db = new CityDatabaseArray();

db.insert("Berlin",50,60);  
db.insert("Tokyo",40 ,70);  
db.insert("Berlin",50,90);  
db.insert("Delhi",20 ,70);  
db.search("Berlin");  
db.printNearby(50,40,20);

}  
}

**Output:**



1. **Linked List Implementation**

public class CityDataBaseLinked {

String name;

int x;

int y;

CityDataBaseLinked next;

public CityDataBaseLinked(String name, int x, int y) {

this.name = name;

this.x = x;

this.y = y;

next = null;

}

public String toString() {

return name + " (" + x + "," + y + ")";

}

}

class CityDatabase {

private CityDataBaseLinked head;

private int size;

public CityDatabase() {

head = null;

size = 0;

}

// Inserts a record with the given name and coordinates

public void insert(String name, int x, int y) {

CityDataBaseLinked newCity = new CityDataBaseLinked(name, x, y);

newCity.next = head;

head = newCity;

size++;

}

// Deletes a record with the given name or coordinates

public void delete(String nameOrCoord) {

if (head == null) {

return;

}

if (head.name.equals(nameOrCoord) || (head.x + "," + head.y).equals(nameOrCoord)) {

head = head.next;

size--;

return;

}

CityDataBaseLinked curr = head;

while (curr.next != null) {

if (curr.next.name.equals(nameOrCoord) || (curr.next.x + "," + curr.next.y).equals(nameOrCoord)) {

curr.next = curr.next.next;

size--;

return;

}

curr = curr.next;

}

}

// Searches for a record with the given name or coordinates and returns its index, or -1 if not found

public CityDataBaseLinked search(String nameOrCoord) {

CityDataBaseLinked curr = head;

while (curr != null) {

if (curr.name.equals(nameOrCoord) || (curr.x + "," + curr.y).equals(nameOrCoord)) {

return curr;

}

curr = curr.next;

}

return null;

}

// Prints all records within the given distance of the specified point

public void printNearby(int x, int y, double distance) {

CityDataBaseLinked curr = head;

while (curr != null) {

double dx = curr.x - x;

double dy = curr.y - y;

double dist = Math.sqrt(dx \* dx + dy \* dy);

if (dist <= distance) {

System.out.println(curr);

}

curr = curr.next;

}

}

// Returns the size of the database

public int size() {

return size;

}

public static void main(String[] args) {

CityDatabase db = new CityDatabase();

db.insert("New York", 0, 0);

db.insert("Los Angeles", 100, 0);

db.insert("Chicago", 50, 50);

CityDataBaseLinked city = db.search("Chicago");

if (city != null) {

System.out.println("Found: " + city);

}

db.delete("New York");

db.printNearby(0, 0, 50);

}

}

**Output:**

****

**Analysis of Algorithms**

1. **Collect running time statistics for each operation in both implementations.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Insert Operation** | **Delete Operation** | **Search Operation** | **printNearby Operation** |
| 0(1) | 0(N) | 0(N) | 0(N) |

Overall, the worst-case time complexity of this implementation is O(n) for most operations, except for the insert operation which has an average-case time complexity of O(1) and a worst-case time complexity of **O(n)** if a resize is needed.

1. **What are your conclusions about the relative advantages and disadvantages of the two implementations?**

When it comes to searching and deleting elements, an implementation using an array is more efficient, whereas implementing a **linked list is more efficient** for inserting elements.

1. **Would storing records on the list in alphabetical order by city name speed any of the operations?**

If records are organized in the list according to the alphabetical order of the city name, it would **speed up the search** operation because the **binary search algorithm** could be used to access the elements.

1. **Would keeping the list in alphabetical order slow any of the operations?**

Inserting new elements into the list while maintaining alphabetical order would **slow down** the insertion operation as new elements would have to be added to the correct position to preserve the alphabetical order.