City Database using Linked List

Aim: To create a city database using linked list.

Theory:
To implement a simple database system for storing and managing city records. The program allows users to insert new city records, delete records by name or coordinates, search for records by name or coordinates, and print all records within a specified distance of a given point. This database system provides basic functionality for managing city data efficiently.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

// Structure to hold city information
typedef struct
{
    char name[100];
    int x;
    int y;
} City;

// Node structure for the unordered list
typedef struct Node
{
    City city;
    struct Node *next;
} Node;

// Function prototypes
Node *insertRecord(Node *head, City city);
Node *deleteRecord(Node *head, const char *name);
Node *deleteRecordByCoordinates(Node *head, int x, int y);
void searchRecordByName(Node *head, const char *name);
void searchRecordByCoordinates(Node *head, int x, int y);
void printRecordsWithinDistance(Node *head, int x, int y, double distance);
double calculateDistance(int x1, int y1, int x2, int y2);

int main()
{
    Node *head = NULL;
    char choice;
    City city;
    char cityName[100];
    int cityX, cityY;
    double distance;

    do
    {
        printf("\nMenu:\n");
```

```
printf("1. Insert a record\n");
printf("2. Delete a record by name\n");
printf("3. Delete a record by coordinates\n");
printf("4. Search a record by name\n");
printf("5. Search a record by coordinates\n");
printf("6. Print records within a given distance of a specified point\n");
printf("7. Exit\n");
printf("Enter your choice: ");
scanf(" %c", &choice);

switch (choice)
{
case '1':
    printf("Enter city name: ");
    scanf("%s", city.name);
    printf("Enter city coordinates (x y): ");
    scanf("%d %d", &city.x, &city.y);
    head = insertRecord(head, city);
    break;
case '2':
    printf("Enter city name to delete: ");
    scanf("%s", cityName);
    head = deleteRecord(head, cityName);
    break;
case '3':
    printf("Enter city coordinates (x y) to delete: ");
    scanf("%d %d", &cityX, &cityY);
    head = deleteRecordByCoordinates(head, cityX, cityY);
    break;
case '4':
    printf("Enter city name to search: ");
    scanf("%s", cityName);
    searchRecordByName(head, cityName);
    break;
case '5':
    printf("Enter city coordinates (x y) to search: ");
    scanf("%d %d", &cityX, &cityY);
    searchRecordByCoordinates(head, cityX, cityY);
    break;
case '6':
    printf("Enter point coordinates (x y): ");
    scanf("%d %d", &cityX, &cityY);
    printf("Enter maximum distance: ");
    scanf("%lf", &distance);
    printRecordsWithinDistance(head, cityX, cityY, distance);
    break;
case '7':
    printf("Exiting program.\n");
    break;
default:
    printf("Invalid choice. Please enter a number from 1 to 7.\n");
}
```

```
    } while (choice != '7');

    return 0;
}

// Function to insert a record into the database
Node *insertRecord(Node *head, City city)
{
    Node *newNode = (Node *)malloc(sizeof(Node));
    newNode->city = city;
    newNode->next = head;
    return newNode;
}

// Function to delete a record by name
Node *deleteRecord(Node *head, const char *name)
{
    Node *current = head;
    Node *prev = NULL;

    while (current != NULL)
    {
        if (strcmp(current->city.name, name) == 0)
        {
            if (prev == NULL)
            {
                head = current->next;
            }
            else
            {
                prev->next = current->next;
            }
            free(current);
            printf("Record with name '%s' deleted.\n", name);
            return head;
        }
        prev = current;
        current = current->next;
    }

    printf("Record with name '%s' not found.\n", name);
    return head;
}

// Function to delete a record by coordinates
Node *deleteRecordByCoordinates(Node *head, int x, int y)
{
    Node *current = head;
    Node *prev = NULL;

    while (current != NULL)
    {
```

```
        if (current->city.x == x && current->city.y == y)
        {
            if (prev == NULL)
            {
                head = current->next;
            }
            else
            {
                prev->next = current->next;
            }
            free(current);
            printf("Record at coordinates (%d, %d) deleted.\n", x, y);
            return head;
        }
        prev = current;
        current = current->next;
    }

    printf("Record at coordinates (%d, %d) not found.\n", x, y);
    return head;
}

// Function to search for a record by name
void searchRecordByName(Node *head, const char *name)
{
    Node *current = head;
    while (current != NULL)
    {
        if (strcmp(current->city.name, name) == 0)
        {
            printf("City found: %s (%d, %d)\n", current->city.name, current->city.x, current->city.y);
            return;
        }
        current = current->next;
    }
    printf("City with name '%s' not found.\n", name);
}

// Function to search for a record by coordinates
void searchRecordByCoordinates(Node *head, int x, int y)
{
    Node *current = head;
    while (current != NULL)
    {
        if (current->city.x == x && current->city.y == y)
        {
            printf("City found: %s (%d, %d)\n", current->city.name, current->city.x, current->city.y);
            return;
        }
        current = current->next;
    }
    printf("City at coordinates (%d, %d) not found.\n", x, y);
```

```
}

// Function to print all records within a given distance of a specified point
void printRecordsWithinDistance(Node *head, int x, int y, double distance)
{
    Node *current = head;
    while (current != NULL)
    {
        double dist = calculateDistance(x, y, current->city.x, current->city.y);
        if (dist <= distance)
        {
            printf("City within %f units: %s (%d, %d)\n", distance, current->city.name, current->city.x,
current->city.y);
        }
        current = current->next;
    }
}

// Function to calculate distance between two points
double calculateDistance(int x1, int y1, int x2, int y2)
{
    return sqrt(pow(x2 - x1, 2) + pow(y2 - y1, 2));
}
```

Output:

```
Menu:
1. Insert a record
2. Delete a record by name
3. Delete a record by coordinates
4. Search a record by name
5. Search a record by coordinates
6. Print records within a given distance of a specified point
7. Exit
Enter your choice: 1
Enter city name: ahmedabad
Enter city coordinates (x y): 1 1

Menu:
1. Insert a record
2. Delete a record by name
3. Delete a record by coordinates
4. Search a record by name
5. Search a record by coordinates
6. Print records within a given distance of a specified point
7. Exit
Enter your choice: 1
Enter city name: gandhinagar
Enter city coordinates (x y): 0 0

Menu:
1. Insert a record
2. Delete a record by name
3. Delete a record by coordinates
4. Search a record by name
5. Search a record by coordinates
6. Print records within a given distance of a specified point
7. Exit
Enter your choice: 3
Enter city coordinates (x y) to delete: 1 1
Record at coordinates (1, 1) deleted.
```

```
Menu:
1. Insert a record
2. Delete a record by name
3. Delete a record by coordinates
4. Search a record by name
5. Search a record by coordinates
6. Print records within a given distance of a specified point
7. Exit
Enter your choice: 4
Enter city name to search: gandhinagar
City found: gandhinagar (0, 0)
```

Algorithm Analysis

Algorithm steps:

1. Initialize an empty linked list to store city records.
2. Display a menu with options for users to interact with the database.
  1. Display menu options:
    - Insert a city record
    - Delete a city record by name
    - Delete a city record by coordinates
    - Search for a city record by name
    - Search for a city record by coordinates
    - Print records within a given distance of a specified point
    - Exit the program
3. Repeat until the user chooses to exit:
  1. Read the user's choice from the menu.
  2. Perform the corresponding operation based on the user's choice:
    - If choice is to insert a city record:
      - Read the city name and coordinates from the user.
      - Create a new city record.
      - Insert the record into the linked list.
    - If choice is to delete a city record by name:
      - Read the city name to delete from the user.
      - Search for the city record by name.
      - If found, delete the record from the linked list.
    - If choice is to delete a city record by coordinates:
      - Read the city coordinates to delete from the user.
      - Search for the city record by coordinates.
      - If found, delete the record from the linked list.
    - If choice is to search for a city record by name:
      - Read the city name to search from the user.
      - Search for the city record by name.
      - If found, display the record.
    - If choice is to search for a city record by coordinates:
      - Read the city coordinates to search from the user.
      - Search for the city record by coordinates.
      - If found, display the record.
    - If choice is to print records within a given distance of a specified point:
      - Read the point coordinates and maximum distance from the user.
      - Iterate through the linked list and print records within the specified distance of the point.

- If choice is to exit:
  - Terminate the program.

4. Upon program termination, free the memory allocated for the linked list.

5. End of algorithm.

Time Complexity:

1. Traversing Linked List for Insertion and Deletion:

- Inserting or deleting a record involves traversing the linked list until the target record is found. This operation takes O(n) time, where n is the number of records in the linked list.

2. Searching for a Record:

- Searching for a record by name or coordinates also requires traversing the linked list. This operation also takes O(n) time in the worst case.

3. Printing Records Within a Distance:

- Printing records within a given distance of a specified point involves traversing the entire linked list and performing distance calculations for each record. This operation takes O(n) time, where n is the number of records in the linked list.

4. Overall:

- All operations in the city database algorithm involve traversing the linked list, resulting in a time complexity of O(n), where n is the number of records in the database.

Space Complexity:

1. Linked List Storage:

- The space required for storing city records in the linked list is proportional to the number of records in the database. Thus, it's O(n), where n is the number of records.

2. Additional Variables:

- The algorithm uses additional variables such as pointers, loop counters, and buffers, which require constant space. Thus, the space complexity for these variables is O(1).

3. Overall:

- Considering the linked list storage and additional variables, the overall space complexity of the city database algorithm is O(n), where n is the number of records in the database.

City Database using Array

Aim: To create a city database using array.

Theory:
This program aims to multiply two numbers represented as linked lists, where each node of the linked list contains a single digit. It utilizes the concept of multiplying two numbers digit by digit, with carry-over handled appropriately.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#define MAX_CITIES 100

// Structure to hold city information
typedef struct {
    char name[100];
    int x;
    int y;
} City;

// Function prototypes
int insertRecord(City cities[], int size, City city);
int deleteRecordByName(City cities[], int size, const char* name);
int deleteRecordByCoordinates(City cities[], int size, int x, int y);
void searchRecordByName(City cities[], int size, const char* name);
void searchRecordByCoordinates(City cities[], int size, int x, int y);
void printRecordsWithinDistance(City cities[], int size, int x, int y, double distance);
double calculateDistance(int x1, int y1, int x2, int y2);

int main() {
    City cities[MAX_CITIES];
    int size = 0;
    char choice;
    City city;
    char cityName[100];
    int cityX, cityY;
    double distance;

    do {
        printf("\nMenu:\n");
        printf("1. Insert a record\n");
        printf("2. Delete a record by name\n");
        printf("3. Delete a record by coordinates\n");
        printf("4. Search a record by name\n");
        printf("5. Search a record by coordinates\n");
        printf("6. Print records within a given distance of a specified point\n");
        printf("7. Exit\n");
        printf("Enter your choice: ");
```

```
        scanf(" %c", &choice);

        switch(choice) {
            case '1':
                printf("Enter city name: ");
                scanf("%s", city.name);
                printf("Enter city coordinates (x y): ");
                scanf("%d %d", &city.x, &city.y);
                size = insertRecord(cities, size, city);
                break;
            case '2':
                printf("Enter city name to delete: ");
                scanf("%s", cityName);
                size = deleteRecordByName(cities, size, cityName);
                break;
            case '3':
                printf("Enter city coordinates (x y) to delete: ");
                scanf("%d %d", &cityX, &cityY);
                size = deleteRecordByCoordinates(cities, size, cityX, cityY);
                break;
            case '4':
                printf("Enter city name to search: ");
                scanf("%s", cityName);
                searchRecordByName(cities, size, cityName);
                break;
            case '5':
                printf("Enter city coordinates (x y) to search: ");
                scanf("%d %d", &cityX, &cityY);
                searchRecordByCoordinates(cities, size, cityX, cityY);
                break;
            case '6':
                printf("Enter point coordinates (x y): ");
                scanf("%d %d", &cityX, &cityY);
                printf("Enter maximum distance: ");
                scanf("%lf", &distance);
                printRecordsWithinDistance(cities, size, cityX, cityY, distance);
                break;
            case '7':
                printf("Exiting program.\n");
                break;
            default:
                printf("Invalid choice. Please enter a number from 1 to 7.\n");
        }
    } while (choice != '7');

    return 0;
}

// Function to insert a record into the database
int insertRecord(City cities[], int size, City city) {
    if (size < MAX_CITIES) {
        cities[size++] = city;
```

```
        printf("Record inserted.\n");
    } else {
        printf("Database full. Cannot insert record.\n");
    }
    return size;
}

// Function to delete a record by name
int deleteRecordByName(City cities[], int size, const char* name) {
    for (int i = 0; i < size; i++) {
        if (strcmp(cities[i].name, name) == 0) {
            for (int j = i; j < size - 1; j++) {
                cities[j] = cities[j + 1];
            }
            printf("Record with name '%s' deleted.\n", name);
            return size - 1;
        }
    }
    printf("Record with name '%s' not found.\n", name);
    return size;
}

// Function to delete a record by coordinates
int deleteRecordByCoordinates(City cities[], int size, int x, int y) {
    for (int i = 0; i < size; i++) {
        if (cities[i].x == x && cities[i].y == y) {
            for (int j = i; j < size - 1; j++) {
                cities[j] = cities[j + 1];
            }
            printf("Record at coordinates (%d, %d) deleted.\n", x, y);
            return size - 1;
        }
    }
    printf("Record at coordinates (%d, %d) not found.\n", x, y);
    return size;
}

// Function to search for a record by name
void searchRecordByName(City cities[], int size, const char* name) {
    for (int i = 0; i < size; i++) {
        if (strcmp(cities[i].name, name) == 0) {
            printf("City found: %s (%d, %d)\n", cities[i].name, cities[i].x, cities[i].y);
            return;
        }
    }
    printf("City with name '%s' not found.\n", name);
}

// Function to search for a record by coordinates
void searchRecordByCoordinates(City cities[], int size, int x, int y) {
    for (int i = 0; i < size; i++) {
        if (cities[i].x == x && cities[i].y == y) {
```

```
            printf("City found: %s (%d, %d)\n", cities[i].name, cities[i].x, cities[i].y);
            return;
        }
    }
    printf("City at coordinates (%d, %d) not found.\n", x, y);
}

// Function to print all records within a given distance of a specified point
void printRecordsWithinDistance(City cities[], int size, int x, int y, double distance) {
    for (int i = 0; i < size; i++) {
        double dist = calculateDistance(x, y, cities[i].x, cities[i].y);
        if (dist <= distance) {
            printf("City within %f units: %s (%d, %d)\n", distance, cities[i].name, cities[i].x, cities[i].y);
        }
    }
}

// Function to calculate distance between two points
double calculateDistance(int x1, int y1, int x2, int y2) {
    return sqrt(pow(x2 - x1, 2) + pow(y2 - y1, 2));
}
```

Output:

```
Menu:
1. Insert a record
2. Delete a record by name
3. Delete a record by coordinates
4. Search a record by name
5. Search a record by coordinates
6. Print records within a given distance of a specified point
7. Exit
Enter your choice: 1
Enter city name: ahmedabad
Enter city coordinates (x y): 1 1
Record inserted.

Menu:
1. Insert a record
2. Delete a record by name
3. Delete a record by coordinates
4. Search a record by name
5. Search a record by coordinates
6. Print records within a given distance of a specified point
7. Exit
Enter your choice: 1
Enter city name: gandhinagar
Enter city coordinates (x y): 0 0
Record inserted.

Menu:
1. Insert a record
2. Delete a record by name
3. Delete a record by coordinates
4. Search a record by name
5. Search a record by coordinates
6. Print records within a given distance of a specified point
7. Exit
Enter your choice: 3
```

```
Menu:
1. Insert a record
2. Delete a record by name
3. Delete a record by coordinates
4. Search a record by name
5. Search a record by coordinates
6. Print records within a given distance of a specified point
7. Exit
Enter your choice: 5
Enter city coordinates (x y) to search: 1 1
City found: ahmedabad (1, 1)
```

Algorithm Analysis

Algorithm steps:

1. Initialize the city database:
- Create an array `cities` to store city records.
- Initialize a variable `size` to keep track of the number of records in the database.

2. Display the menu and process user input:
- Display a menu with options for various operations on the city database.
- Read the user's choice from the menu.
- Based on the user's choice, perform the corresponding operation.

3. Insert a city record:
- Read the city name, x-coordinate, and y-coordinate from the user.
- Create a new city record with the provided information.
- Check if the database is full.
- If not full, add the new city record to the `cities` array and increment `size`.
- If the database is full, display an error message.

4. Delete a city record by name:
- Read the city name to delete from the user.
- Search for the city record with the given name in the `cities` array.
- If found, remove the record by shifting all subsequent records to fill the gap.
- Decrement `size` to reflect the removal.
- If not found, display an appropriate message.

5. Delete a city record by coordinates:
- Read the x-coordinate and y-coordinate of the city to delete from the user.
- Search for the city record with the given coordinates in the `cities` array.
- If found, remove the record by shifting all subsequent records to fill the gap.
- Decrement `size` to reflect the removal.
- If not found, display an appropriate message.

6. Search for a city record by name:
- Read the city name to search from the user.
- Search for the city record with the given name in the `cities` array.
- If found, display the city information (name, x-coordinate, y-coordinate).
- If not found, display an appropriate message.

7. Search for a city record by coordinates:
- Read the x-coordinate and y-coordinate of the city to search from the user.
- Search for the city record with the given coordinates in the `cities` array.
- If found, display the city information (name, x-coordinate, y-coordinate).
- If not found, display an appropriate message.

8. Print records within a given distance of a specified point:
- Read the x-coordinate, y-coordinate, and maximum distance from the user.
- Iterate through the `cities` array.
- For each city, calculate the distance from the specified point using the `calculateDistance` function.
- If the distance is less than or equal to the maximum distance, print the city information.

9. Exit the program:
- Display a message indicating program termination.

10. End of algorithm.

Time Complexity:

1. Inserting a Record:
- In the worst case, inserting a record involves iterating through the entire `cities` array once to find an empty slot. This operation has a time complexity of O(n), where n is the maximum number of cities (MAX_CITIES).

2. Deleting a Record by Name or Coordinates:
- Deleting a record by name or coordinates involves iterating through the `cities` array once to find the record to delete. In the worst case, this operation also has a time complexity of O(n).

3. Searching for a Record by Name or Coordinates:
- Searching for a record by name or coordinates requires iterating through the `cities` array once to find the desired record. This operation has a time complexity of O(n) in the worst case.

4. Printing Records Within a Given Distance of a Specified Point:
- Printing records within a given distance of a specified point involves iterating through the `cities` array once and calculating the distance for each record. This operation has a time complexity of O(n) in the worst case.

5. Overall:
- Considering the above operations, the overall time complexity of the city database program is O(n), where n is the number of records in the database.

Space Complexity:

1. Storage for City Records:
- The space required to store city records in the `cities` array is proportional to the maximum number of cities (MAX_CITIES). Thus, the space complexity for storing city records is O(MAX_CITIES).

2. Additional Variables:
- The program uses additional variables such as loop counters and temporary variables, which require constant space. Therefore, the space complexity for additional variables is O(1).

3. Overall:

- Considering the storage for city records and additional variables, the overall space complexity of the city database program is O(MAX_CITIES).