ALGORITHMS AND DATA STRUCTURE – MIDTERM REPORT

1. Objective

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 database should allow records to be inserted, deleted by name or coordinate, and searched by name or coordinate. Another operation that should be supported is to print 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.
Collect running time statistics for each operation in both implementations.
What are your conclusions about the relative advantages and disadvantages of the two implementations?

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

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

2. Analysis

- In this midterm, student needs to create a city database by using array and linked list (which will be made in unordered list).
- The record list must include name of the municipality (city) and its longitude and latitude (coordinates).
- These inputs type is different.
- There are some requirement functions such as insert, delete, search, print.
- The array will store data at contiguous memory locations.
- The linked list will store data at an arbitrary memory locations.
- Furthermore, user can provide a point and a distance to find the city located in point that giving the same distance.
- The running time calculation is a requirement after completing program. Therefore, student could compare two implementations (pros and cons)

3. Results (Code) and running time records

These programs (codes) and running time records can be found by this link:
https://github.com/ndsang001/Algorithms_data_structure_midterm.git

4. Experiences and description

- The task was really difficult and also needing a lot of researches.

- I have got stuck and felt stress

- However, I tried to complete this and got a quite good outcome.

- *The array-based-list implementation:*

- The “structures” (a user-defined data) have been used to store these different inputs in an array structure.

```
struct dataType{
    string name;
    int x;
    int y;
};
```

- A “class municipality” was created, which includes main functions (Insert, delete, search and check). There is no “Constructor” generated.

- Functions have been made in turn. Firstly, we need to get the data from user so the “Insert” function need to be written. These data would be verified before saving for making sure there is no existed city name or coordinates.

```
if(!city.checkExisted(x1, y1, name1, size, data1)){
    city.insertCityFront(size, x1, y1, name1, data1);
    }
```

- When inserting at the front of array, the whole array must move to right to reserve the first storage for the new one *(O(n))*. Meanwhile, inserting at an arbitrary point also take *O(n)* time to run.

- Besides, delete function will make the array move to left and decrease array’s size. Delete function needs more conditions: isEmpty and checkExisted. These conditions would prove array itself not empty and also the expected city existed.

```
if(isEmpty(a) == true){
    cout << "This city list is empty!!!" << endl;
  } else if(checkExist(a, name1, data1) == -1){
    cout << "This city name does not exist!!!" << endl;
  } else {
    int pos = checkExist(a, name1, data1);
    for(int i = pos; i < a; ++i){
```

```
        data1[i] = data1[i+1];
      }
      a--;
      cout << "This city name " << name1 << " have been
deleted!" << endl;
    }
```

- Similarly, the logic for searching and printing this city list array is the same for other functions.

```
+ if(isEmpty(a) == true){
      cout << "This city list is empty!!!" << endl;
    } else if(checkExist(a, name1, data1) == -1){
      cout << "This city name does not exist!!!" << endl;
    }else {
      int pos = checkExist(a, name1, data1);
      cout << " This city " << name1 << " is located in position
" << pos + 1 << endl;
      cout << "With the coordinates: " << " x = " <<
data1[pos].x << " and y = " << data1[pos].y << endl;
    }
```

- For checking which city meets the input conditions (distance and point), the data will be passed to check function and then be calculated in each term of array. This results in printing all city having the same distance from its coordinates to the provided point.

```
+ void check(int a, int x1, int y1, int z, dataType *data1){
   if(isEmpty(a) == true){
     cout << "This city list is empty!!!" << endl;
   } else {
     int b ,c ,d, e = 0;
     for(int i = 0; i < a; ++i){
       b = data1[i].x;
       c = data1[i].y;
       d = sqrt(pow((x1-b),2) + pow((y1-c),2));
```

```
        if(d == z){
          cout << "This city is in the distance: " <<
data1[i].name << endl;
          e++;
        }
      }
      if(e == 0){
        cout << "There is no city in the distance" << endl;
      }
    }
```

- In "int main()" function, the municipality object "city" was generated. Moreover, the "data1[100]" object was also assigned. A menu option will be printed out and user now can choose option or exit by entering 0 (which runs by do while and switch case operation).

  *** In this program, the data inputted by user would be mainly passed to the function by *reference and pointer.*

- *The linked list implementation*

- There were two "classes" created: "class Node{}" and "class cityList{}"

- In the Node class, there are 4 data member and a constructer with parameters was made for passing input data to a "node" of linked list.

```
+ Node(double a, double b, string d){
    x = a;
    y = b;
    name = d;
  }
```

- Another constructor "Node" assigned the pointer "next" to NULL.

  → So now we could create a node (but not exactly linked list)

- Inside the cityList (class), there is only one pointer assigned to "head" (data member) of linked list.

```
+ Node* head;
```

- This "head" pointer would only point to the object (class node) and hence there is a constructor to assign "head" pointing to "n" (node).

- Insert, delete, search, print and check functions are also put here (*class cityList*).
- With the insert (at the end) function, we will assign the new data to the "next" pointer (After some condition such as cityExist, coordinatesExist and head = NULL)

```
+ Node *ptr = head;
      while(ptr->next != NULL){
      ptr = ptr->next;
      }
      ptr->next = n;
      cout << "City has been added!" << endl;
```

- It was not quite complicated in "insert" at the front. The "new node" will point to the "first node" and then "head" will point to "new node".

```
+ n->next = head;
        head = n;
        cout << "City has been added!" << endl;
```

- "Delete" function goes along with some checking condition, the ptr (pointer) will move in turn until the "ptr->next" (current pointer) meets the name of the expected city. This leads to assigning the temp (NULL pointer) with the current pointer. Therefore, now the previous pointer could point to the next one of "temp" pointer.

```
+ Node* temp = NULL;
          Node* prevptr = head;
          Node* currentptr = head->next;
          while(currentptr != NULL){
            if(currentptr->name == d){
              temp = currentptr;
              currentptr = NULL;
            } else {
              prevptr = prevptr->next;
```

```
        currentptr = currentptr->next;
      }
    }
    if(temp != NULL){
      prevptr->next = temp->next;
      cout << "This city name is unlinked: " << d << endl;
    } else {
      cout << "There is no city name: " << d << endl;
    }
```

- The logic for pointer was applied for whole program so the printing and search were in the same concept.
- For checking which city meets the input conditions (distance and point), the data will be passed to the “check” function then a while loop is used to check the condition.

```
+ Node *currentptr = head;

    while(currentptr != NULL){
      i = currentptr->x;
      j = currentptr->y;
      z = sqrt(pow((i-a),2) + pow((j-b),2));
      if(z == c){
     cout << "This city is in the distance: " <<
     currentptr->name << endl;
      }
        currentptr = currentptr->next;
```

- In “int main()” function, the cityList object “city” was generated. A menu option will be printed out and user now can choose option or exit by entering 0 (which runs by do while and switch case operation).

5. Comparation array and linked list implementation

- It is easier and faster to access an element by using array while linked list take a linear time.

- The data will be store contiguously in memory by using array while it is random in linked list.

- The memory cost for each element is lower in using array because linked list node includes next and previous references.

- The insert and delete functions will be performed better in a linked list implementation and these functions would take time in an array.

- It is flexible for linked list to expand size. On the contrary, array has a fixed size.

6. Application of alphabetical order

- A sorted list by alphabetical order could help the "*search*" function faster (in some cases) because we now just need to look up until meeting the greater character (alphabet).

- However, it is not essential to make a sorted list just only for the easy searching because it is comparatively for program (computer) to sort an array or linked list.

- The "*Insert*" function now take linear time because it needs to search the place based on alphabet table and then move the rest list to right.