Ngoc Nguyen NITK17K - 1700062

Mid-term Project

Question
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?

Github link
https://github.com/NgocNguyenGit/Algorithms-and-Data-Structures

Objectives
Array:
- Unordered lists
- Elements:
  - Name of the city (a string of arbitrary length)
  - The coordinates (integer): x and y
- My approach in this project is to create a simple and fixed size database.

Operations (functions) supported by the array and can be called by the user in the interface:
- **Insertion:** method insert takes a record and inserts it into the city database
- **Deletion:** The remove method is similar to find, except that it also deletes the record returned from the city database
- **Search:** Method find takes a key value and returns some record from the city database whose key matches the one provided
- **Distance Evaluation:** Give distance of a specified point and print all records within it

Implementation
1) Array-based list
In this implementation, the user is allowed to enter input in the interface, which means there is no sample/default database available in the array.
- Insert operation:

unsortedArray.h, ***O(1)***	aList.h, ***O(1)***

`// Insert an element: append to list`
`void insert(const Key&k, const E& e) {`
`  KVpair<Key,E> temp(k, e);`
`  list->append(temp);`
`}`
	`void append(const E& it) {  // Append "it"`
	`  Assert(listSize < maxSize, "List capacity exceeded");`
	`  listArray[listSize++] = it;`
	`}`

- In main, ***O(n)***

```
void insert_function(){
    //insert
    for (int i = 0; i < newSize+1; i++){
        cout<<"Enter x y name: ";
        cin>>x>>y>>name;
        foo[i] = new CityDB(x,y,name);

        xDict.insert(foo[i]->getX(), foo[i]);
        yDict.insert(foo[i]->getY(), foo[i]);
        nameDict.insert(foo[i]->getName(), foo[i]);
        newSize++;
        cout<<"Continue? [y/n]: ";
        cin>>isCont;

        if (isCont == 'n'){
            break;
        }
    }
}
```

- Find operation:

unsortedArray.h, ***O(n)***	aList.h, ***O(1)***

`// Find "k" using sequential search`
`E find(const Key& k) const {`
`  for(list->moveToStart(); list->currPos() < list->length(); list->next()) {`
`  KVpair<Key,E> temp = list->getValue();`
`  if (k == temp.key()){`
`  return temp.value();`
`  }`
`  }`
`  return NULL; // "k" does not appear in dictionary`
`}`
	`void prev() { if (curr != 0) curr--; }  // Back up`
	`void next() { if (curr < listSize) curr++; } // Next`
	
	`void moveToStart() { curr = 0; }  // Reset position`
	`void moveToEnd() { curr = listSize; }  // Set at end`
	
	`// Return list size`
	`int length() const  { return listSize; }`
	
	`// Return current position`
	`int currPos() const { return curr; }`

- In main, ***O(1)***

```
void find_function(int findWhat){
    cout<<"Search function...."<<endl;
    cout<<"1-x      2-y     3-name"<<endl;
    cout<<"Search what? [1/2/3]: ";
    cin>>findWhat;
    if (findWhat == 1){
        //x
        cout<<"Find x: ";
        cin>>x;
        findx = xDict.find(x);
        if (findx != NULL){
            cout << findx;
        } else {
            cout << "NULL X COORDINATE.\n";
        }
    } else if (findWhat == 2){
        //y
        cout<<"Find y: ";
        cin>>y;
        findy = yDict.find(y);
        if (findy != NULL){
            cout << findy;
        } else {
            cout << "NULL Y COORDINATE.\n";
        }
    } else {
        //name
        cout<<"Find name: ";
        cin>>name;
        findname = nameDict.find(name);
        if (findname != NULL){
            cout << findname;
        } else {
            cout << "NULL NAME.\n";
        }
    }
}
```

- Delete operation:

unsortedArray.h, ***O(1)***	aList.h, ***O(n)***

```
// Use sequential search to find the element to remove
E remove(const Key& k) {
    E temp = find(k); // "find" will set list position
    if(temp != NULL)
        list->remove();
    return temp;
}
```

```
// Remove and return the current element.
E remove() {
    Assert((curr>=0) && (curr < listSize), "No element");
    E it = listArray[curr];                // Copy the element
    for(int i=curr; i<listSize-1; i++){    // Shift them down
        listArray[i] = listArray[i+1];
    }
    listSize--;                            // Decrement size
    return it;
}
```

- Distance evaluation:
    - In main, ***O(n)***

```
void distance_function(){
    cout<<"Distance function...."<<endl;
    cout<<"Enter distance: ";
    cin>>dist;
    cout<<"Enter x: ";
    cin>>x;
    cout<<"Enter y: ";
    cin>>y;
    for (int i = 0; i < newSize; i++){
        dist_result = (float)sqrt(powf((float)x-foo[i]->getX(),2)+powf((float)y-foo[i]->getY(),2));
        if (dist_result <= dist){
            cout<<foo[i]->getX()<<", "<<foo[i]->getY()<<", "<<foo[i]->getName()<<endl;
        }
    }
}
```

An sample of output with time taken by all the functions in the array-based list implementation is **71867568 microseconds**.

```
Enter x y name: 1111 22222 Saigon
Continue? [y/n]: y
Enter x y name: -333333 4444 Helsinki
Continue? [y/n]: y
Enter x y name: 123455 543221 Singapore
Continue? [y/n]: n
******************************
Print city database....
1111, 22222, Saigon
-333333, 4444, Helsinki
123455, 543221, Singapore
******************************
Search function....
1-x      2-y     3-name
Search what? [1/2/3]: 3
Find name: Helsinki
-333333, 4444, Helsinki
******************************
Delete function....
1-x      2-y     3-name
Delete what? [1/2/3]: 2
Delete y: 22222
Deleted 1111, 22222, Saigon
******************************
Distance function....
Enter distance: 10000
Enter x: 123455
Enter y: 543221
123455, 543221, Singapore
******************************
Time taken by function: 71867568 microseconds
Program ended with exit code: 0
```

2) Linked list

In this implementation, the user is not allowed to enter input in the interface, which means there are samples/default records database available in the array.

- Insert operation:

- Insert operation, ***O(1)***

```
void UnsortedType::insertRecord(int x, int y, string name){ //O(1)
    Node* temp = new Node;
    temp->dataX = x;
    temp->dataY = y;
    temp->dataName = name;

    if (start == NULL)
    {
        start = temp;
        end = temp;
    }
    else
    {
        end->next = temp;
        end = temp;
    }
    length++;
}
```

- Delete operation, ***O(n)***

Delete coordinates

```
void UnsortedType::del(int x, int y){
    Node *current = new Node;
    Node *previous = new Node;
    current = start;
    if (start->dataX == x && start->dataY == y)
    {
        start = start->next;
    }
    else if (end->dataX == x && end->dataY == y)
    {
        for (int i = 0; i < length - 1; i++)
        {
            previous = current;
            current = current->next;
        }
        end = previous;
        previous->next = NULL;
        length--;
    }
    else
    {
        for (int i = 0; i < length; i++)
        {
            if (current->dataX == x && current->dataY == y)
            {
                previous->next = current->next;
                length--;
                break;
            }
            else
            {
                previous = current;
                current = current->next;
            }
        }
    }
}
```

Delete name

```
void UnsortedType::del(string name)
{
    Node *current = new Node;
    Node *previous = new Node;
    current = start;
    if (start->dataName == name)
    {
        start = start->next;
    }
    else if (end->dataName == name)
    {
        for (int i = 0; i < length - 1; i++)
        {
            previous = current;
            current = current->next;
        }
        end = previous;
        previous->next = NULL;
        length--;
    }
    else
    {
        for (int i = 0; i < length; i++)
        {
            if (current->dataName == name)
            {
                previous->next = current->next;
                length--;
                break;
            }
            else
            {
                previous = current;
                current = current->next;
            }
        }
    }
}
```

- Search

Search coordinates

```
void UnsortedType::search(int x, int y)
{
    Node *current = new Node;
    Node *previous = new Node;
    current = start;
    for (int i = 0; i < length; i++)
    {
        if (current->dataX == x && current->dataY == y)
        {
            cout << "City: " << current->dataName << ", x: " << current->dataX << ", y: " << current->dataY << endl;
        }
        previous = current;
        current = current->next;
    }
}
```

Search name	
Search city within distance	

```
void UnsortedType::search(string name)
{
    Node *current = new Node;
    Node *previous = new Node;
    current = start;
    for (int i = 0; i < length; i++)
    {
        if (current->dataName == name)
        {
            cout << "City: " << current->dataName << ", x: " << current->dataX << ", y: " << current->dataY << endl;
        }
        previous = current;
        current = current->next;
    }
}
```

```
void UnsortedType::search(int distance, int x, int y)
{
    Node *current = new Node;
    Node *previous = new Node;
    current = start;
    for (int i = 0; i < length; i++)
    {
        if (sqrt(pow(current->dataX - x,2) + pow(current->dataY - y, 2)) <= distance)
        {
            cout << "City: " << current->dataName << ", x: " << current->dataX << ", y: " << current->dataY << endl;
        }
        previous = current;
        current = current->next;
    }
}
```

An sample of output with time taken by all the functions in the array-based list implementation is **276 microseconds**.

```
City: Saigon, x: 5, y: 6
5, 6, Saigon
7, 30, Hanoi
Time taken by function: 276 microseconds
Program ended with exit code: 0
```

In main

```
#include <iostream>
#include <chrono>
#include "linkedList.h"
using namespace std;
using namespace chrono;
int main(int argc, const char * argv[]) {
    UnsortedType ust;
    //sample database
    // Get starting timepoint
    auto start = high_resolution_clock::now();
    ust.insertRecord(5,6, "Saigon");
    ust.insertRecord(7,30, "Hanoi");
    ust.insertRecord(8,20000, "Singapore");
    ust.insertRecord(-19,5, "Helsinki");
    ust.del(-19, 5);                    //delete coordinate
    ust.del("Singapore");               //delete name
    ust.search(10, 5,6);                //distance evaluate
    ust.show();                         //print all
    // Get ending timepoint
    auto stop = high_resolution_clock::now();

    // Get duration. Substart timepoints to
    // get durarion. To cast it to proper unit
    // use duration cast method
    auto duration = duration_cast<microseconds>(stop - start);

    cout << "Time taken by function: "<< duration.count() << " microseconds" << endl;
    return 0;
}
```

Program analysis

Collect running time statistics for each operation in both implementation:

- Before storing records on the list in alphabetical order by city name:
    - Array-based list: insert is a constant-time operation; find and delete require O(n) time in the average and worst cases.
    - Linked list: the cost of the functions should be the same asymptotically.
- After storing records on the list in alphabetical order by city name and give a conclusion on which operation will be speeded up and which will be slowed down
    - As in some references, it is supposed that the find operation can be speeded up by using a binary search O(logn). The user cannot be permitted to control

where elements get inserted, nor append elements onto the list. It means that the insert operation here has O(n) time.

- Conclusion: the more find operations are used than insert, the more it might be worth using a sorted list to implement the city database.

Analyze the relative advantages and disadvantages of the two implementations

Array-based list	Linked list
Advantages: - There is no wasted space for an individual element. When the array for the array-based list is completely filled, there is no storage overhead. Therefore, it will then be more space efficient, by a constant factor, than the linked implementation. Disadvantages: - The size must be redetermined before the array can be allocated. It means that array-based lists cannot grow beyond their predetermined size. - The amount of space required by a array-based list implementation is Ω(n) but can be bigger. Conclusion: Array-based lists are generally more space efficient when the user knows in advance approximately how large the list will become. They are faster for random access by position which operations always take O(1) time.	Advantages: - Linked list only need space for the objects actually on the list and there is no limit to the number of elements on a linked list, as long as there is free-store memory available. - The amount of space required by a linked list is O(n). Disadvantages: - Lined lists require that an extra pointer be added to every list node. If the element size is small, the overhead for links can be a significant fraction of the total storage. Conclusion: Linked lists are more space efficient when implementing lists whose number of elements varies widely or is unknown. In this project, singly linked lists have no explicit access to the previous element, and access by position requires that I march down the list from the front to the specified position. This require O(n) time in the average.

The concept of key and comparable objects

Search key:

- Name of city
- Coordinates x and y