Date: 24/02/2021

Practical 4:

2CSDE75 - Advanced Data Structures

Name: Shrey Viradiya

Roll No: 18BCE259

Aim:

Skip list structure are used to retrieve the data faster. Implement the structure up to third level. Show the effect of insert and delete operation.

Code:

Prac4_SkipList.cpp

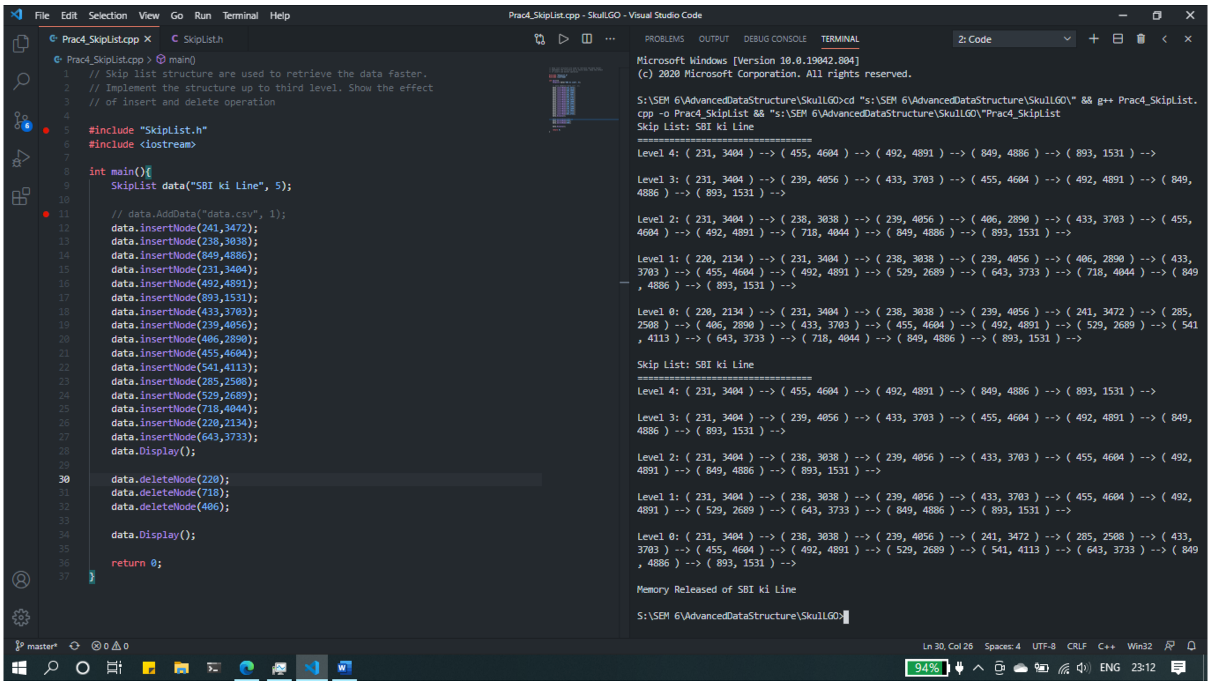
```cpp
// Skip list structure are used to retrieve the data faster.
// Implement the structure up to third level. Show the effect
// of insert and delete operation

#include "SkipList.h"
#include <iostream>

int main(){
    using namespace std;
    SkipList data("SBI ki Line", 5);

    // data.AddData("data.csv", 1);
    data.insertNode(241,3472);
    data.insertNode(238,3038);
    data.insertNode(849,4886);
    data.insertNode(231,3404);
    data.insertNode(492,4891);
    data.insertNode(893,1531);
    data.insertNode(433,3703);
    data.insertNode(239,4056);
    data.insertNode(406,2890);
    data.insertNode(455,4604);
    data.insertNode(541,4113);
    data.insertNode(285,2508);
    data.insertNode(529,2689);
    data.insertNode(718,4044);
    data.insertNode(220,2134);
    data.insertNode(643,3733);
    data.Display();

    data.deleteNode(220);
    data.deleteNode(718);
    data.deleteNode(406);

    data.Display();

    cout << data.Search(643) << endl;
    cout << data.Search(718) << endl;
    cout << data.Search(231) << endl;

    return 0;
}
```

SkipList.h

```cpp
#include <iostream>
#include <cstring>
#include <cstdlib>
#include <stdexcept> // std::runtime_error
#include <sstream>
#include <fstream>
#include <string>

class SkipListNode{
        int key;
        int object;
        int levels;
    public:
        SkipListNode **forwards;

        SkipListNode() = delete;
        SkipListNode(int level, int k, int o){
            forwards = new SkipListNode*[level + 1];
            for (int i = 0; i <= level; i++) forwards[i] = nullptr;
            key = k;
            object = o;
            levels = level + 1;
        }
        ~SkipListNode(){
            delete [] forwards;
        }
        int getKey(){
            return key;
        }
        int getObject(){
            return object;
        }
        int getLevels(){
            return levels;
        }
};

class SkipList{
    public:
        char name[50];
        const int level_limit;
        SkipListNode **InitialLinks;

        SkipList(const char nameinput[50], int limit);
        ~SkipList();
        int Search(int key);
        void insertNode(int key, int object);
        void deleteNode(int key);
        void Display();
        void AddData(std::string filename, int isHeading);
};

SkipList::SkipList(const char nameinput[50], const int limit) : level_limit(limit){
    strcpy(name, nameinput);
    InitialLinks = new SkipListNode*[level_limit];
    for (int i = 0; i < level_limit; i++) InitialLinks[i] = nullptr;
}

SkipList::~SkipList(){
    delete [] InitialLinks;
    using namespace std;
    cout << "Memory Released of " << name << endl;
}

void SkipList::insertNode(int key, int object){

    // Keep a list of search path for updating
    SkipListNode **update = new SkipListNode* [level_limit];
    for (int i = 0; i < level_limit; i++) update[i] = nullptr;
    int upperLimit = 0;
    SkipListNode *iterator = nullptr;
    for (int i = level_limit - 1; i >= 0; i--)
    {
        if (InitialLinks[i] != nullptr && InitialLinks[i]->getKey() < key)
        {
            iterator = InitialLinks[i];
            upperLimit = i;
            break;
        }
        update[i] = InitialLinks[i];
    }

    if (iterator == nullptr)
    {
        int randomNumber = (int)((rand()*level_limit)/RAND_MAX);
        SkipListNode *newNode = new SkipListNode(randomNumber, key, object);

        for (int i = 0; i <= randomNumber; i++)
        {
            if (update[i] != nullptr){
                newNode->forwards[i] = update[i];
                InitialLinks[i] = newNode;
            }
            else{
                newNode->forwards[i] = nullptr;
                InitialLinks[i] = newNode;
            }
        }
    }
    else{
        for (int i = upperLimit; i >= 0; i--)
        {
            while (iterator->forwards[i] != nullptr && iterator->forwards[i]-
>getKey() < key)
            {
                iterator = iterator->forwards[i];
            }
            update[i] = iterator;
        }
        int randomNumber = rand()%level_limit;
        SkipListNode *newNode = new SkipListNode(randomNumber, key, object);

        for (int i = 0; i <= randomNumber; i++)
        {
            if (update[i] != nullptr){
                newNode->forwards[i] = update[i]->forwards[i];
                (*update[i]).forwards[i] = newNode;
            }
            else{
                newNode->forwards[i] = nullptr;
                InitialLinks[i] = newNode;
            }
        }
    }
}

void SkipList::Display(){
    using namespace std;
    cout << "Skip List: " << name << endl;
    cout << "=================================" << endl;

    for (int i = level_limit -1; i >= 0; i--)
    {
        SkipListNode *iterator = InitialLinks[i];

        cout << "Level " << i << ": ";
        while (iterator != nullptr)
        {
            cout << "( "<< iterator->getKey() << ", " << iterator-
>getObject() << " ) --> ";
            iterator = iterator->forwards[i];
        }
        cout << endl;
        cout << endl;
    }
}

void SkipList::AddData(std::string filename, int isHeading){
    using namespace std;
    // working with csv in CPP
    // https://www.gormanalysis.com/blog/reading-and-writing-csv-files-with-cpp/

    ifstream myFile(filename);
    // if(!myFile.is_open()) throw runtime_error("Could not open file");

    string line, word;
    int val;

    if (isHeading)  getline(myFile, line);

    // Read data, line by line
    while(getline(myFile, line))
    {
        // Create a stringstream of the current line
        stringstream ss(line);
        pair<int, int> data;

        // add the column data
        // of a row to a pair
        getline(ss, word, ',');
        data.first = stoi(word);

        getline(ss, word, ',');
        data.second = stoi(word);

        this->insertNode(data.first, data.second);
    }

    // Close file
    myFile.close();
}

void SkipList::deleteNode(int key){
    // Keep a list of search path for updating
    SkipListNode **update = new SkipListNode* [level_limit];
    for (int i = 0; i < level_limit; i++) update[i] = nullptr;
    int upperLimit = 0;
    SkipListNode *iterator = nullptr;
    for (int i = level_limit - 1; i >= 0; i--)
    {
        if (InitialLinks[i] != nullptr && InitialLinks[i]->getKey() < key)
```

```cpp
        {
            iterator = InitialLinks[i];
            upperLimit = i;
            break;
        }
        update[i] = InitialLinks[i];
    }

    if(iterator == nullptr){
        iterator = InitialLinks[0];
        if (iterator->getKey() == key)
        {
            for (int i = 0; i < iterator->getLevels(); i++)
            {
                InitialLinks[i] = iterator->forwards[i];
                iterator->forwards[i] = nullptr;
            }
            delete iterator;
        }
    }
    else{
        for (int i = upperLimit; i >= 0; i--)
        {
            while (iterator->forwards[i] != nullptr && iterator->forwards[i]-
>getKey() < key)
            {
                iterator = iterator->forwards[i];
            }
            update[i] = iterator;
        }
        iterator = iterator->forwards[0];
        if (iterator->getKey() != key)
        {
            return;
        }
        else{
            for (int i = 0; i < iterator->getLevels(); i++)
            {
                (*update[i]).forwards[i] = iterator->forwards[i];
                iterator->forwards[i] = nullptr;
            }
            delete iterator;
        }
    }
}

int SkipList::Search(int key){
    int upperLimit = 0;
    SkipListNode *iterator = nullptr;
    for (int i = level_limit - 1; i >= 0; i--)
    {
        if (InitialLinks[i] != nullptr && InitialLinks[i]->getKey() < key)
        {
            iterator = InitialLinks[i];
            upperLimit = i;
            break;
        }
    }

    if (iterator == nullptr)
    {
        iterator = InitialLinks[0];
        if(iterator->getKey() == key){
            return iterator->getObject();
        }
        return -1;
    }
    else{
        for (int i = upperLimit; i >= 0; i--)
        {
            while (iterator->forwards[i] != nullptr && iterator->forwards[i]-
>getKey() < key)
            {
                iterator = iterator->forwards[i];
            }
        }
        iterator = iterator->forwards[0];

        if (iterator->getKey() == key)
        {
            return iterator->getObject();
        }
        return -1;
    }
}
```

Snapshot of the output:



```cpp
// Skip list structure are used to retrieve the data faster.
// Implement the structure up to third level. Show the effect
// of insert and delete operation

#include "SkipList.h"
#include <iostream>

int main(){
    SkipList data("SBI ki Line", 5);

    // data.AddData("data.csv", 1);
    data.insertNode(241,3472);
    data.insertNode(238,3038);
    data.insertNode(849,4886);
    data.insertNode(231,3404);
    data.insertNode(492,4891);
    data.insertNode(893,1531);
    data.insertNode(433,3703);
    data.insertNode(239,4056);
    data.insertNode(406,2890);
    data.insertNode(455,4604);
    data.insertNode(541,4113);
    data.insertNode(285,2508);
    data.insertNode(529,2689);
    data.insertNode(718,4044);
    data.insertNode(220,2134);
    data.insertNode(643,3733);
    data.Display();

    data.deleteNode(220);
    data.deleteNode(718);
    data.deleteNode(406);

    data.Display();

    return 0;
}
```

```
Microsoft Windows [Version 10.0.19042.804]
(c) 2020 Microsoft Corporation. All rights reserved.

S:\SEM 6\AdvancedDataStructure\SkulLGO>cd "s:\SEM 6\AdvancedDataStructure\SkulLGO\" && g++ Prac4_SkipList.
cpp -o Prac4_SkipList && "s:\SEM 6\AdvancedDataStructure\SkulLGO\"Prac4_SkipList
Skip List: SBI ki Line
=================================
Level 4: ( 231, 3404 ) --> ( 455, 4604 ) --> ( 492, 4891 ) --> ( 849, 4886 ) --> ( 893, 1531 ) -->

Level 3: ( 231, 3404 ) --> ( 239, 4056 ) --> ( 433, 3703 ) --> ( 455, 4604 ) --> ( 492, 4891 ) --> ( 849,
4886 ) --> ( 893, 1531 ) -->

Level 2: ( 231, 3404 ) --> ( 238, 3038 ) --> ( 239, 4056 ) --> ( 406, 2890 ) --> ( 433, 3703 ) --> ( 455,
4604 ) --> ( 492, 4891 ) --> ( 718, 4044 ) --> ( 849, 4886 ) --> ( 893, 1531 ) -->

Level 1: ( 220, 2134 ) --> ( 231, 3404 ) --> ( 238, 3038 ) --> ( 239, 4056 ) --> ( 406, 2890 ) --> ( 433,
3703 ) --> ( 455, 4604 ) --> ( 492, 4891 ) --> ( 529, 2689 ) --> ( 643, 3733 ) --> ( 718, 4044 ) --> ( 849
, 4886 ) --> ( 893, 1531 ) -->

Level 0: ( 220, 2134 ) --> ( 231, 3404 ) --> ( 238, 3038 ) --> ( 239, 4056 ) --> ( 241, 3472 ) --> ( 285,
2508 ) --> ( 406, 2890 ) --> ( 433, 3703 ) --> ( 455, 4604 ) --> ( 492, 4891 ) --> ( 529, 2689 ) --> ( 541
, 4113 ) --> ( 643, 3733 ) --> ( 718, 4044 ) --> ( 849, 4886 ) --> ( 893, 1531 ) -->

Skip List: SBI ki Line
=================================
Level 4: ( 231, 3404 ) --> ( 455, 4604 ) --> ( 492, 4891 ) --> ( 849, 4886 ) --> ( 893, 1531 ) -->

Level 3: ( 231, 3404 ) --> ( 239, 4056 ) --> ( 433, 3703 ) --> ( 455, 4604 ) --> ( 492, 4891 ) --> ( 849,
4886 ) --> ( 893, 1531 ) -->

Level 2: ( 231, 3404 ) --> ( 238, 3038 ) --> ( 239, 4056 ) --> ( 433, 3703 ) --> ( 455, 4604 ) --> ( 492,
4891 ) --> ( 849, 4886 ) --> ( 893, 1531 ) -->

Level 1: ( 231, 3404 ) --> ( 238, 3038 ) --> ( 239, 4056 ) --> ( 433, 3703 ) --> ( 455, 4604 ) --> ( 492,
4891 ) --> ( 529, 2689 ) --> ( 643, 3733 ) --> ( 849, 4886 ) --> ( 893, 1531 ) -->

Level 0: ( 231, 3404 ) --> ( 238, 3038 ) --> ( 239, 4056 ) --> ( 241, 3472 ) --> ( 285, 2508 ) --> ( 433,
3703 ) --> ( 455, 4604 ) --> ( 492, 4891 ) --> ( 529, 2689 ) --> ( 541, 4113 ) --> ( 643, 3733 ) --> ( 849
, 4886 ) --> ( 893, 1531 ) -->

Memory Released of SBI ki Line

S:\SEM 6\AdvancedDataStructure\SkulLGO>
```

Conclusion:

With Skip List, we can retrieve data faster than linked list. Implementation is harder than linked list and more error prone, but it is worth it if working with a long list.