TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI

KHOA CÔNG NGHỆ THÔNG TIN

BÀI TẬP LỚN MÔN HỌC

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Giảng viên hướng dẫn: TS. Hoàng Văn Thông

Sinh viên thực hiện: Phạm Duy Lợi

Mã SV: 222601120

Lớp: CNTT – VA1

Hà Nội, tháng 11 năm 2023

Mục lục

BÀI 1 – BÀI SỐ 35 TRONG DANH SÁCH BÀI TẬP

I. Đề bài

Cài đặt lớp danh sách liên kết kép mẫu, bổ sung phương thức sắp xếp các phần tử theo thứ tự tăng dần hoặc dần tùy thuộc vào tham số.

Ứng dụng giải bài toán sau:

Một buổi họp mặt đại gia đình nhân dịp cụ già Ted tròn 100 tuổi, người ta muốn sắp xếp con cháu của cụ theo thứ tự từ tuổi cao xuống thấp. Giả sử ta có thông tin về giấy khai sinh của từng người đó. Mỗi giấy khai sinh chỉ viết ba thông tin đơn giản gồm: Tên người cha, Tên người con, Tuổi của người cha lúc sinh con.

Hãy giúp đại gia đình trên tính ra tuổi của từng người con cháu cụ Ted và viết ra danh sách theo thứ tự từ tuổi cao xuống thấp.

Đầu vào:

Dòng đầu ghi số bộ test (không quá 100). Với mỗi bộ test:

- Dòng đầu tiên ghi số X (0<X<100) là số người con cháu cần sắp xếp.
- Tiếp theo là X dòng, mỗi dòng ghi thông tin về một giấy khai sinh của từng người (thứ tự ngẫu nhiên) gồm 3 thành phần, mỗi thành phần cách nhau một khoảng trống:
  - o Tên người cha: không quá 20 ký tự và không chứa khoảng trống
  - o Tên người con: không quá 20 ký tự và không chứa khoảng trống
  - o Tuổi của người cha khi sinh con: 1 số nguyên dương, không quá 100.

Đầu ra:

- Với mỗi bộ test, in ra màn hình thứ tự bộ test (xem thêm trong bộ test ví dụ), sau đó lần lượt là từng người trong danh sách tuổi từ cao xuống thấp (không tính cụ Ted). Mỗi người viết ra hai thông tin: tên, một khoảng trống rồi đến tuổi của người đó.
- Nếu hai người có cùng tuổi thì xếp theo thứ tự từ điển.

Ví dụ:

Input:

2
1
Ted Bill 25
4
Ray James 40
James Beelzebub 17
Ray Mark 75
Ted Ray 20

Output:

DATASET 1
Bill 75
DATASET 2
Ray 80
James 40
Beelzebub 23
Mark 5

II. Phân tích bài toán

- Bài toán yêu cầu cài đặt dánh sách liên kết kép, bổ sung phương thức sắp xếp, sau đó ứng dụng giải bài toán tìm tuổi của các con cháu của Ted và sắp xếp theo thứ tự giảm dần, nếu cùng tuổi thì sắp xếp tên theo thứ tự từ điển
- Với danh sách liên kết kép, mỗi phần tử sẽ liên kết với phần tử đứng trước và sau nó trong danh sách. Mỗi phần tử trong danh sách (node) gồm biến lưu dữ liệu và 2 con trỏ liên kết tới phần tử trước và sau nó.
- Các phương thức xây dựng:
  - Tạo một danh sách các con cháu của Ted
  - Thêm con cháu vào danh sách
  - Sắp xếp danh sách theo sao cho tên người cha luôn xuất hiện trước người con
  - Tính tuổi của các con cháu của Ted
  - Sắp xếp các con cháu theo tuổi và theo thứ tự từ điển
  - Hiển thị danh sách ra màn hình
- Các lớp xây dựng trong bài:
  - Node
  - Doublelist
- Phương thức trong Node:
  - GetNext: trả về địa chỉ Node tiếp theo
  - GetPre: trả về địa chỉ Node trước đó
  - SetNext: gắn giá trị cho thuộc tính Next
  - SetPre: gắn giá trị cho thuộc tính Pre
  - Setdad(F ch): gắn giá trị tên cha cho thuộc tính F
  - Setson(S c): gắn giá trị tên con cho thuộc tính S
  - Setage(A t): gắn giá trị tuổi cha khi sinh con cho thuộc tính A
  - Getdad: trả về địa chỉ của tên cha
  - Getson: trả về địa chỉ của tên con
  - Getage: trả về tuổi
- Phương thức trong Doublelist:
  - Doublelist: khởi tạo danh sách liên kết kép rỗng ban đầu
  - Node pushback: thêm Node vào cuối danh sách
  - Node pushfront: thêm Node vào đầu danh sách
  - Void remove: xóa Node
  - Void insert: chèn Node
  - Void swapnode: hoán đổi giá trị 2 Node
  - Void push: Nhập danh sách con cháu của Ted
  - Void sortdad: Sắp xếp danh sách sao cho tên cha luôn xuất hiện trước tên con
  - Void calcul: tính tuổi các con cháu
  - Void sort: sắp xếp theo tuổi và theo thứ tự từ điển

- Void printf: in ra màn hình

III. Cài đặt các lớp và hàm main bằng C++

- Đầu tiên xây dựng class Node:

```
template <class F,class S, class A>
class Node{
    private:
        F cha;
        S con;
        A tuoi;
        Node<F,S,A> *next;
        Node<F,S,A> *pre;
    public:
        Node(){
            next = 0;
            pre = 0;
        }
        Node<F,S,A> *getNext(){return next;}
        Node<F,S,A> *getPre(){return pre;}
        void setNext(Node<F,S,A> *p){next = p;}
        void setPre(Node<F,S,A> *p){pre = p;}
        F getdad(){return cha;}
        void setdad(F ch){cha = ch;}
        S getson(){return con;}
        void setson(S c){con = c;}
        A getage(){return tuoi;}
        void setage(A t){tuoi = t;}
};
```

- Xây dựng class Doublelist và import Node đã cài đặt vào:

```
class Doublelist {
    private:
        Node<F,S,A> *header;
        Node<F,S,A> *trailer;

    public:
        Doublelist(){…
        Node<F,S,A> *pushback(F ch, S c, A t){…
        Node<F,S,A> *pushfront(F ch, S c, A t){…
        void push(int n){…
        void remove(Node<F, S, A>* node) {…
        void insert(Node<F,S,A> *node1, Node<F,S,A> *node2){…
        void sortdad(){…
        void calcul(){…
        void swapNodes(Node<F, S, A>* node1, Node<F, S, A>* node2) {…
        void quicksort(Node<F, S, A>* low, Node<F, S, A>* high) {…

        Node<F, S, A>* partition(Node<F, S, A>* low, Node<F, S, A>* high) {…
        void sort(){…
        void printf(){…
};
```

- Hàm main:

```
int main() {
    int t;
    cout <<"Nhap so truong hop: ";
    cin >> t;

    for (int test = 1; test <= t; ++test) {
        int n;
        cout<<"Nhap so con chau truong hop thu " << test <<": ";
        cin >> n;
        Doublelist<string, string, int> familyList;
        familyList.push(n);
        familyList.sortdad();
        familyList.calcul();
        familyList.sort();
        cout << "DATASET " << test << endl;
        familyList.printf();
    }
}
```

IV. Phân tích thời gian chạy của các phương thức có trong lớp:

- Các phương thức trong class Node đều có thời gian chạy O(1)

- Push: Thời gian chạy O(n)

```
void push(int n){
    for (int i = 0; i < n; ++i) {
        string father, child;
        int age;
        cin >> father >> child >> age;
        if(father == "Ted") pushfront(father,child,age);
        else pushback(father, child, age);
    }
}
```

- Sortdad: Thời gian chạy O(n^2)

```
void sortdad(){
    for(Node<F,S,A> *i=header;i!=nullptr;i=i->getNext()){
        for(Node<F,S,A> *j=i->getNext();j!=nullptr;j=j->getNext()){
            if(j->getdad() == i->getson())
                insert(i,j);
        }
    }
}
```

- Calcul: Thời gian chạy O(n^2)

```
void calcul(){
    Node<F,S,A> *current = header;
    while (current != nullptr) {
        if (current->getdad() == "Ted") {
            current->setage(100 - current->getage());
        } else {
        // Tìm tuổi của cha của người con
        Node<string, string, int> *father = header;
        while (father != nullptr) {
            if (father->getson() == current->getdad()) {
                current->setage(father->getage() - current->getage());
                break;
            }
            father = father->getNext();
        }
    }
    current = current->getNext();
    }
}
```

- Sort: sử dụng thuật toán QuickSort, thời gian chạy O(nlogn)

```
void quicksort(Node<F, S, A>* low, Node<F, S, A>* high) {
if (low != nullptr && high != nullptr && low != high && low->getPre() != high) {
    Node<F, S, A>* pivot = partition(low, high);

    quicksort(low, pivot->getPre());
    quicksort(pivot->getNext(), high);
    }
}

Node<F, S, A>* partition(Node<F, S, A>* low, Node<F, S, A>* high) {
    A pivotAge = high->getage();
    Node<F, S, A>* i = low->getPre();

    for (Node<F, S, A>* j = low; j != high; j = j->getNext()) {
        if (j->getage() > pivotAge || (j->getage() == pivotAge && j->getson() < high->getson())) {
            i = (i == nullptr) ? low : i->getNext();
            swapNodes(i, j);
        }
    }

    i = (i == nullptr) ? low : i->getNext();
    swapNodes(i, high);
    return i;
}
```

- Printf: Thời gian chạy O(n)

```
    void printf(){
        Node<F,S,A> *current = header;
        while (current != nullptr) {

            cout << current->getson() << " " << current->getage() << endl;

        current = current->getNext();
    }
    }
```

BÀI 2 – BÀI SỐ 7 TRONG DANH SÁCH BÀI TẬP

I. Đề bài

1. Xây dựng lớp biểu diễn một bảng băm sử dụng một trong những hàm băm đã học

o Sử dụng lớp bảng băm để xây dựng lớp biểu diễn từ điển Anh Việt. Mỗi phần tử của từ điển là một cặp (tiếng anh, nghĩa tiếng việt). Với các phương thức:

a. Nạp từ điển từ file vào bảng băm

b. Tìm kiếm bằng phương pháp tìm kiếm trên bảng băm

c. Sửa đổi các từ

d. Thêm từ mới

e. Xóa từ

f. Lưu từ điển vào file

2. Xây dựng chương trình có các chức năng

a. Nạp từ điển từ file (có thể nạp tự động khi chạy chương trình)

b. Tra từ điển (nhập vào một từ, hiển thị nghĩa của nó nếu có trong từ điển)

c. Sửa đổi từ

d. Thêm từ mới

e. Xóa từ

f. Lưu từ điển vào file

II. Phân tích bài toán

- Bài toán yêu cầu xây dựng lớp biểu diễn bảng băm và sử dụng hàm băm đã học để xây dựng chương trình từ điển có thể thêm từ, tra từ, sửa đổi từ,xóa từ, lưu vào file.
- Bảng băm là 1 cấu trúc mà khi người dùng thực hiện truy xuất một phần từ qua khóa thì nó sẽ được ánh xạ vào thông qua hàm băm.
- Các phương thức xây dựng:
  - Tạo bảng băm chứa từ điển anh việt
  - Đọc file có sẵn
  - Lưu vào file
  - Tra từ điển
  - Thêm từ
  - Xóa từ
  - Hiển thị ra màn hình từ điển
  - Hàm băm (sử dụng phương pháp djb2) -Daniel J.Bernstein
- Các lớp xây dựng trong bài:
  - Node
  - SingleList
  - Hashtable
  - Dictionary
- Các phương thức trong Node:
  - Getkey: trả lại địa chỉ của từ tiếng anh
  - Setkey: gắn giá trị từ tiếng anh vào Keys
  - getNext: trả lại địa chỉ Node kế tiếp
  - setnext: gắn địa chỉ cho thuộc tính Next
  - getElem: trả lại địa chỉ của nghĩa tiếng việt
  - setElem: gắn giá trị của từ tiếng việt vào Elem
- Các phương thức trong SingleList:
  - Node getHead: trả về địa chỉ của Node đầu tiên
  - SingleList : khởi tạo danh sách liên kết đơn rỗng ban đầu

- Long size: trả về số lượng các phần từ trong danh sách
- Int isEmpty: kiểm tra xem danh sách có rỗng không
- Node pushback: thêm Node vào cuối danh sách
- Void replace: thay thế Node cũ bằng Node mới
- Void remove: xóa Node
- Node getNode: trả về giá trị của Node cần tìm

- Các phương thức trong Hashtable:
  - Hashtable: tạo bảng băm và tạo danh sách liên kết để lưu trữ các phần tử trong bảng băm
  - ~hashtable: giải phóng bộ nhớ, xóa các danh sách liên kết
  - Add: thêm phần tử vào bảng băm, nếu tồn tại thì trả về Null còn không thì trả về phần tử vừa thêm
  - Remove: xóa phần từ khỏi bảng băm
  - Find: Tìm kiếm phần tử trong bảng băm
  - Count: đếm số lượng phần tử trong bảng băm
  - Getsize: trả về kích thước của bảng băm
  - Gettable: trả về bảng băm chứa các danh sách liên kết
- Các phương thức trong Dictionary:
  - Dictionary: khởi tạo từ điển
  - LoadFromFile: đọc file và thêm dữ liệu vào từ điển
  - SaveToFile: lưu dữ liệu vào File
  - Lookup: tìm kiếm nghĩa trong từ điển
  - Modify: Sửa đổi nghĩa của từ đã có trong từ điển
  - Remove: xóa từ khỏi từ điển
  - Display: hiển thị danh sách từ điển ra màn hình
  - Hash: hàm băm

III. Cài đặt các lớp và hàm main vào C++

- Đầu tiên xây dựng class Node:

```
class Node{
    private:
        Keys key;
        T elem;
        Node<Keys,T> *next;
    public:
        Node() { next = 0; }
        Keys getKey(){ return key;}
        void setKey(Keys k){ key = k;}
        Node<Keys,T>  *getNext(){return next;}
        void setNext(Node<Keys,T> *p){next = p;}
        T getElem(){return elem;}
        void setElem(T e){elem = e;}
    };
```

- Xây dựng class SingleList và import Node đã cài đặt vào:

```
class SingleList{
private:
    Node<Keys,T> *header;
    Node<Keys,T> *trailer;
    long sz;
public:
    Node<Keys, T> *getHead() {return header;}
    SingleList(){…
    long size(){return sz;}

    int isEmpty(){return sz==0;}
    Node<Keys,T>* pushback(Keys k,T e)  //Chen them mot node vao cuoi danh sach…
    void replace(Node<Keys,T>* p,Keys k, T e)…
    void remove(Node<Keys,T> *p)         // Loai bo node do con tro p tro toi…
    Node<Keys,T>* getNode(Keys k)  //tim kiem tuan tu…
};
```

- Xây dựng class Hashtable vào import SingleList vào:

```
class Hashtable
{
    private:
        int m;
        SingleList<Keys,T> *table;
    public:
        Hashtable(int n);
        ~Hashtable(){
           delete[] table;
        }
        Node<Keys,T> *Add(Keys key, T e,  int (*hash)(Keys,int));
        void Remove(Keys key, int (*hash)(Keys,int));
        Node<Keys,T> *Find(Keys key, int (*hash)(Keys,int));
        //bool Contains(Keys key, int (*hash)(Keys,int));
        int Count();
        int getSize() {
           return m;
        }
        SingleList<Keys, T>* getTable(){
           return table;
        }

};
```

- Xây dựng class Dictionary và import Hashtable vào:

```
class Dictionary {
private:
    Hashtable<string, string> hashtable;

public:
    Dictionary(int size) : hashtable(size) {}

    void LoadFromFile(const string &filename) {···

    void SaveToFile(const string &filename) {···

    string Lookup(const string &word) {···

    void Modify(const string &word, const string &meaning) {···

    void AddWord(const string &word, const string &meaning) {···

    void RemoveWord(const string &word) {···

    void Display() {···

    static int Hash(string key, int size) {···


};
```

- Hàm Main:

```
int main() {
    Dictionary dictionary(100);

    dictionary.LoadFromFile("dictionary.txt");

    while (true) {
        cout << "--------------Menu---------------\n";
        cout << "a. Lookup word\n";
        cout << "b. Modify word\n";
        cout << "c. Add word\n";
        cout << "d. Remove word\n";
        cout << "e. Display dictionary\n";
        cout << "f. Save dictionary to file\n";
        cout << "h. Exit\n";
        cout << "---------------------------------\n";
        cout << "Choose an option: ";

        char choice;
        cin >> choice;

        if (choice == 'a') {
            cout << "Enter a word to lookup: ";
            string word;
            cin >> word;
            cout <<"Meaning of "<<word<<": "<< dictionary.Lookup(word) << endl;
        } else if (choice == 'b') {
            cout << "Enter a word to modify: ";
            string word;
            cin >> word;
            cout << "Enter the new meaning: ";
            string meaning;
            cin.ignore();
            getline(cin, meaning);
            dictionary.Modify(word, meaning);
```

```
    } else if (choice == 'c') {
        cout << "Enter a new word: ";
        string word;
        cin >> word;
        cout << "Enter the meaning: ";
        string meaning;
        cin.ignore();
        getline(cin, meaning);
        dictionary.AddWord(word, meaning);
    } else if (choice == 'd') {
        cout << "Enter a word to remove: ";
        string word;
        cin >> word;
        dictionary.RemoveWord(word);
    } else if (choice == 'e') {
        dictionary.Display();
    } else if (choice == 'f') {
        dictionary.SaveToFile("dictionary.txt");
        cout << "Dictionary saved to file." << endl;
    } else if (choice == 'h') {
        break;
    } else {
        cout << "Invalid choice. Please try again." << endl;
    }
}
```

IV. Phân tích thời gian chạy của các phương thức có trong lớp

- Các phương thức trong Class Node và Class SingleList có thời gian chạy O(1)
- Class Hashtable:
  - Hastable: thời gian chạy O(n)

```
Hashtable<Keys,T>::Hashtable(int n)
{
        m = n;
        table=  new SingleList<Keys,T>[m];
}
```

  - Add: thời gian chay O(1)

```
Node<Keys,T>* Hashtable<Keys,T>::Add(Keys key, T e, int (*hash)(Keys,int))
{
    Node<Keys,T> *p;
    int h = hash(key,m); //hash la mot con tro ham
    p = table[h].getNode(key);
    if(p==NULL)
        return table[h].pushback(key,e);
    else
        return NULL;
}
```

- Remove and Find: thời gian chạy O(1)

```
void Hashtable<Keys,T>::Remove(Keys key, int (*hash)(Keys,int))
{
    int h = hash(key,m);
    Node<Keys,T> *p;
    p = table[h].getNode(key);
    if(p!=NULL)
        table[h].remove(p);
}
template<class Keys,class T>
Node<Keys,T> * Hashtable<Keys,T>::Find(Keys key, int (*hash)(Keys,int))
{
    int h = hash(key,m);
    return  table[h].getNode(key);
}
```

- Count: thời gian chạy O(m+n)

```
int Hashtable<Keys,T>::Count()
{
     int t = 0;
     for(int i=0;i<m;i++)
        t = t + table[i].size();
     return t;
}
```

- Class Dictionary:
  - Dictionary: thời gian chạy O(n)
  - LoadFromFile: thời gian chạy O(n)

```
    void LoadFromFile(const string &filename) {
        ifstream file(filename);
        if (file.is_open()) {
            string line;
            while (getline(file, line)) {
                size_t pos = line.find(" ");
                if (pos != string::npos) {
                    string word = line.substr(0, pos);
                    string meaning = line.substr(pos + 1);
                    hashtable.Add(word, meaning, Hash);
                }
            }
            file.close();
        }
    }
```

- SavetoFile: thời gian chạy O(m+n)

```
void SaveToFile(const string &filename) {
    ofstream file(filename);
    if (file.is_open()) {
        for (int i = 0; i < hashtable.getSize(); i++) {
            Node<string, string> *node = hashtable.getTable()[i].getHead();
            while (node != nullptr) {
                file << node->getKey() << "  " << node->getElem() << endl;
                node = node->getNext();
            }
        }
        file.close();
    }
}
```

- Lookup, Modify, Addword, RemoveWord: thời gian chạy: O(1) đến O(n) tùy thuộc vào cách phân phối trong hashtable

```
string Lookup(const string &word) {
    Node<string, string> *node = hashtable.Find(word, Hash);
    if (node != nullptr) {
        return node->getElem();
    }
    return "Word not found in the dictionary.";
}

void Modify(const string &word, const string &meaning) {
    hashtable.Remove(word, Hash);
    hashtable.Add(word, meaning, Hash);
}

void AddWord(const string &word, const string &meaning) {
    hashtable.Add(word, meaning, Hash);
}

void RemoveWord(const string &word) {
    hashtable.Remove(word, Hash);
}
```

- Display: thời gian chạy O(m+n)

```
void Display() {
    cout << "-----Dictionary-----" << endl;
    for (int i = 0; i < hashtable.getSize(); i++) {
        Node<string, string> *node = hashtable.getTable()[i].getHead();
        while (node != nullptr) {
            cout << node->getKey() << " -" << node->getElem() << endl;
            node = node->getNext();
        }
    }
    cout << "--------------------" << endl;
}
```