**TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI**

**VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG**

**Ảnh có chứa văn bản, áp phích, Phông chữ, Đồ họa

Mô tả được tạo tự động**

**BÁO CÁO TIẾN ĐỘ TUẦN**

**Môn học: Project 1**

***Chủ đề: Ứng dụng thuật toán tìm kiếm***

**Giáo viên hướng dẫn: Lê Bá Vui**

**Sinh viên thực hiện: *Lê Minh Triết***

**Hà Nội - 2024**

Contents

[1. Công việc đã hoàn thành trong tuần 3](#_Toc186281687)

[1.1. Thêm minh họa cho các thuật toán Search bằng animation 3](#_Toc186281688)

[2. Dự kiến công việc tuần sau 4](#_Toc186281689)

[3. Source code 4](#_Toc186281690)

Chương 1. Giới thiệu đề tài và sản phẩm

1. Vấn đề cần giải quyết

Vấn đề cần giải quyết là xây dựng một ứng dụng có thể minh họa cách các thuật toán tìm kiếm cơ bản hoạt động.

2. Mục tiêu và phạm vi của đề tài

Trên cơ sở vấn đề cần giải quyết, mục tiêu của bài tập là tạo ra một ứng dụng (gọi là SearchApp) có thể thực hiện các công việc như: tìm kiếm phần tử của một dãy số theo 1 thuật toán nào đó, so sánh các thuật toán dựa theo thời gian thực hiện, minh họa bằng animation các thuật toán tìm kiếm.

Ứng dụng được phát triển trên nền tảng desktop.

3. Định hướng giải pháp

Trên cơ sở xác định mục tiêu và phạm vi của đề tài, em quyết định thiết kế ứng dụng theo mô hình hướng đối tượng, sử dụng ngôn ngữ C++ và lập trình frontend bằng Qt. Chi tiết sẽ được nhắc tới trong phần sau.

3. Các chức năng chính của ứng dụng

Trong phần này, em sẽ trình bày một vài chức năng chính của ứng dụng, bao gồm

* Triển khai được các thuật toán tìm kiếm các phần tử trên một dãy số nguyên, đảm bảo tính đúng đắn khi tìm kiếm.
* So sánh hiệu quả của các thuật toán tìm kiếm dựa trên bộ đếm thời gian
* Minh họa các thuật toán tìm kiếm bằng animation.
* Thực hiện nhập xuất dữ liệu qua bàn phím, qua file.
* Có thể tạo các bộ sinh số.

Chương 2. Cơ sở lý thuyết và công nghệ sử dụng

1. Cơ sở lý thuyết

1.1. Thuật toán Linear Search

Thuật toán Linear Search chạy đúng với bất kỳ mảng nào.

Ý tưởng chính của thuật toán này là bắt đầu từ phần tử đầu tiên của mảng, so sánh phần tử ở vị trí hiện tại. Nếu bằng nhau thì tìm được phần tử cần tìm, nếu không thì tăng chỉ số hiện tại lên 1. Trong trường hợp duyệt tới vị trí vượt qua vị trí cuối cùng thì chứng tỏ phần tử cần tìm không xuất hiện trong mảng.

Độ phức tạp thuật toán này là O(n).

1.2. Thuật toán Binary Search

Thuật toán Binary Search yêu cầu mảng đã được sắp xếp.

Ý tưởng của thuật toán là chia để trị. Nó sẽ giữ hai chỉ số *left* và *right* để tìm kiếm, ban đầu đó là 2 chỉ số đầu và cuối của mảng.

Ở mỗi bước, nó sẽ so sánh phần tử ở giữa (*mid*) với phần tử cần tìm kiếm. Nếu bằng nhau thì tìm được phần tử cần tìm kiếm, nếu phần tử ở giữa nhỏ hơn thì ta chỉ cần tìm kiếm ở phần bên phải, tức là cho *left = mid + 1*, còn không ta chỉ cần tìm kiếm ở mảng bên trái, tức là cho *right = mid – 1* (nếu dãy giảm dần thì ngược lại). Thuật toán sẽ kết thúc khi tại một bước nào đó thì phần tử ở vị trí *mid* bằng phần tử cần tìm kiếm hoặc *left > right*.

Độ phức tạp trung bình của thuật toán này là O(log n).

1.3. Thuật toán Ternary Search

Thuật toán Ternary Search yêu cầu mảng đã được sắp xếp.

Ý tưởng của thuật toán này là chia để trị giống hết Binary Search. Tuy nhiên thay vì chỉ so sánh với phần tử ở giữa và chia mảng làm 2 phần, TernarySearch chia mảng thành 3 phần và so sánh với 2 phần tử chia, sau đó thực hiện tìm kiếm trên các thành phần thu nhỏ.

Độ phức tạp trung bình của thuật toán này là O(log3 n).

1.4. Thuật toán Jump Search

Thuật toán Jump Search yêu cầu mảng đã được sắp xếp.

Ý tưởng chính của thuật toán là chia mảng thành các mảng con gồm các phần tử liên tiếp có độ dài bằng nhau (độ dài mỗi mảng con chính là ). Sau đó, ta nhảy qua từng khối cho tới khi đạt được giá trị lớn hơn hoặc bằng giá trị cần tìm. Cuối cùng ta thực hiện tìm kiếm tuần tự trong khối tương ứng.

Độ phức tạp trung bình của thuật toán này là O().

1.5. Thuật toán Exponential Search

Thuật toán Exponential Search yêu cầu mảng đã được sắp xếp.

Thuật toán được triển khai như sau: Bắt đầu từ phần tử đầu tiên và kiểm tra theo cấp số mũ (tức là kiểm tra ở các vị trí 1, 2, 4, 8, 16,...) cho đến khi vượt quá hoặc gặp phần tử cần tìm hoặc đạt đến phần cuối của mảng. Sau khi xác định được khoảng chứa phần tử cần tìm (ví dụ từ 2k-1 đến 2k), thực hiện tìm kiếm nhị phân trong khoảng này.

Độ phức tạp của thuật toán này trong tình huống tệ nhất là O(log n).

1.6. Thuật toán Interpolation Search

Thuật toán Interpolation Search yêu cầu mảng đã được sắp xếp, đặc biệt hiệu quả trong trường hợp các phần tử của mảng được phân bố đều (tức khoảng cách giữa mọi cặp phần tử liên tiếp nhau không sai khác quá nhiều).

Ý tưởng chính của thuật toán này cũng là chia để trị. Tuy nhiên khác Binary Search, nó không so sánh phần tử ở chính giữa mà vị trí được so sánh được tính bằng công thức nội suy.

Giả sử ta đang tìm kiếm trên một mảng con từ vị trí *low* tới vị trí *high* trên mảng chính *arr*. Khi đó vị trí *pos* của phần tử cần so sánh với phần tử cần tìm kiếm *x* được tính bởi công thức

Sau đó dựa vào tính nhỏ hơn, lớn hơn hay bằng nhau của phần tử vị trí *pos* với phần tử *x* mà ta quyết định duyệt dãy con trái, dãy con phải như Binary Search.

Độ phức tạp trung bình của thuật toán là O(log log n), nhưng trong trường hợp tệ nhất sẽ là O(n) nếu các phần tử phân bố không đều.

1.7. Thuật toán Fibonacci Search

Thuật toán Fibonacci Search yêu cầu mảng đã được sắp xếp.

Fibonacci Search hoạt động dựa trên nguyên lý chia mảng theo tỷ lệ Fibonacci. Dãy Fibonacci là một dãy số trong đó mỗi số là tổng của hai số trước đó: 0, 1, 1, 2, 3, 5, 8, 13, 21, ... Thuật toán sử dụng các số Fibonacci để chia mảng thành các phần nhỏ hơn và liên tục thu hẹp phạm vi tìm kiếm cho đến khi tìm thấy phần tử cần tìm hoặc xác nhận rằng phần tử không tồn tại trong mảng.

Độ phức tạp trung bình của thuật toán này là O(log n).

1.8. Thuật toán Hashing Search

Thuật toán này có ý tưởng khác biệt so với tất cả thuật toán trên. Nó giảm thời gian tìm kiếm bằng cách tăng không gian lưu trữ.

Cụ thể, thuật toán sẽ sử dụng một hàm băm để ánh xạ giá trị cần tìm vào một chỉ số trong bảng băm. Để chèn một phần tử, tính toán giá trị băm và lưu trữ phần tử tại vị trí tương ứng trong bảng băm. Để tìm kiếm một phần tử, tính toán giá trị băm và kiểm tra phần tử tại vị trí tương ứng.

Độ phức tạp trung bình của thuật toán là O(1), trong khi tồi nhất là O(n), khi xảy ra quá nhiều xung đột.

1.9. So sánh về mặt lý thuyết độ phức tạp về thời gian của các thuật toán tìm kiếm

Sau đây là bảng so sánh độ phức tạp về thời gian của các thuật toán tìm kiếm, sử dụng tiệm cận Big O và *n* là số phần tử của mảng cần tìm kiếm.

Bảng 1. Bảng so sánh độ phức tạp các thuật toán tìm kiếm theo tiệm cận Big O

|  |  |  |
| --- | --- | --- |
| **Thuật toán** | **Trung bình** | **Tệ nhất** |
| Linear Search |  |  |
| Binary Search |  |  |
| Ternary Search |  |  |
| Jump Search |  |  |
| Exponential Search |  |  |
| Interpolation Search |  |  |
| Fibonacci Search |  |  |
| Hashing Search | 1 |  |

2. Công nghệ sử dụng

Thứ nhất, về ngôn ngữ lập trình, em sử dụng ngôn ngữ C++ do tính phù hợp của nó kể cả việc thiết kế các thuật toán và thiết kế hướng đối tượng.

Thứ hai, về phần lập trình giao diện, em sử dụng Qt Creator để tạo một giao diện đơn giản, thân thiện.

Qt là một framework phần mềm đa nền tảng rất phổ biến trong phát triển ứng dụng. Được phát triển bởi Qt Company (trước đây là Trolltech), Qt cung cấp một bộ công cụ mạnh mẽ cho việc xây dựng các ứng dụng GUI (giao diện đồ họa người dùng), cũng như các ứng dụng không giao diện, đa nền tảng.

Chương 3. Thiết kế và xây dựng ứng dụng

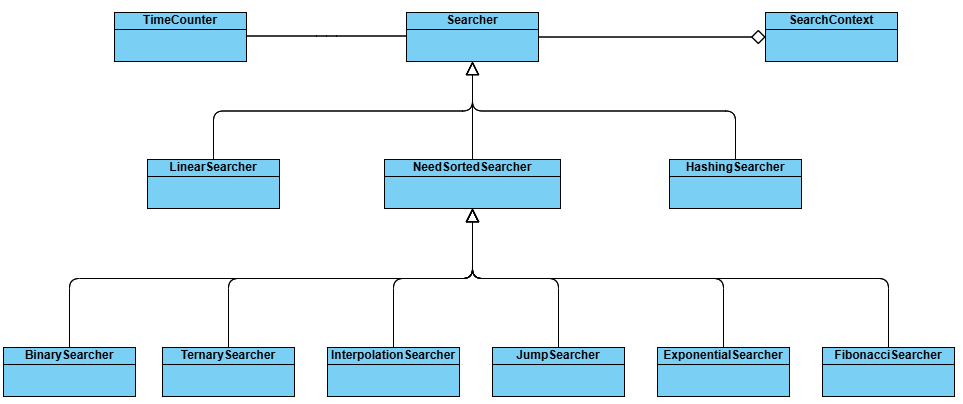
1. Thiết kế tổng quan

Trong phần này, em sẽ trình bày thiết kế tổng quan của ứng dụng. Ứng dụng sẽ được chia thành 3 module chính để phát triển, đó là: Search Module (cài đặt các thuật toán tìm kiếm trên dãy số nguyên), IO Module (cài đặt các phương thức để truyền dữ liệu đầu vào và ra) và Visualize Module (cài đặt những animation để minh họa các thuật toán).

Ngoài ra, để tăng tính module hóa, em chia mã nguồn thành các file header (.h) và các file mã nguồn (.cpp) để dễ dàng quản lý.

Sau đây là các biểu đồ class, miêu tả tổng quan các class được chứa trong mỗi module và mối quan hệ giữa chúng.

1.1. Thiết kế Search Module



Hình 1. Biểu đồ Class cho Search Module

Trong biểu đồ trên, class TimeCounter sẽ hỗ trợ xây dựng bộ đếm thời gian để đo lường các thuật toán tìm kiếm. Searcher là class cha của mọi thuật toán tìm kiếm, trong số các con của nó thì có NeedSortedSearcher đại diện cho các thuật toán tìm kiếm hoạt động trên dãy được sắp xếp. Ngoài ra, class SearchContext được tạo ra để quyết định Searcher nào được sử dụng tại thời điểm run-time.

1.2. Thiết kế IO Module

Ảnh có chứa ảnh chụp màn hình, biểu đồ, hàng, Hình chữ nhật

Mô tả được tạo tự động

Hình 2. Biểu đồ Class cho IO Modulo

Module này đại diện cho 3 phương thức mà người dùng có thể thực hiện để truyền dữ liệu đầu vào, đó là: vào bằng bàn phím, qua field (thực hiện qua class Reader); vào bằng file (thực hiện bằng class FileIO); vào bằng bộ sinh số (thực hiện qua class ArrGenerator).

Hiện em thiết kế 3 loại bộ sinh số, đó là: tăng dần liên tiếp (AscendingArrGenerator), giảm dần liên tiếp (DescendingArrGenerator), ngẫu nhiên (RandomizeArrGenerator).

1.3. Thiết kế Visualizer Module

Ảnh có chứa hàng, biểu đồ, Song song, Hình chữ nhật

Mô tả được tạo tự động

Hình 3. Biểu đồ class cho Visualize Module

Do trong module minh họa các thuật toán này, giả thiết các dãy đầu vào đều là tăng dần (tức là dãy mẫu minh họa đã được xác định trước) nên mọi visualizer của các thuật toán sắp xếp đều chỉ cần kế thừa các phương thức visualize cơ bản từ SearchVisualizer.

2. Thiết kế chi tiết lớp

Trong phần này, em sẽ trình bày về thiết kế chi tiếp của 3 lớp cơ bản nhất của ứng dụng, đó là Searcher, NeedSortedSearcher và một lớp ví dụ về 1 thuật toán tìm kiếm cụ thể là BinarySearcher.

Đầu tiên là thiết kế cho lớp Searcher như sau:

Thứ nhất là các prototype được định nghĩa trong file header Searcher.h

class Searcher {

protected:

vector<int> arr;

vector<TimeCounter> timeCounters;

public:

vector<int> getArr();

virtual void setArr(vector<int> arr);

double getTotalSearchTime();

double getTotalBuildTime();

void addTimeCounter(TimeCounter newTimeCounter);

TimeCounter getLastTimeCounter();

virtual void clear();

virtual string getAlgorithm() = 0;

virtual int search(int value) = 0;

Searcher();

Searcher(vector<int> arr);

};

Và thiết kế chi tiết trong Searcher.cpp

vector<int> Searcher::getArr() {

return arr;

}

void Searcher::setArr(vector<int> arr) {

this->arr = arr;

}

double Searcher::getTotalSearchTime() {

double totalSearchTime = 0;

for (TimeCounter timeCounter: timeCounters) {

totalSearchTime += timeCounter.getSearchTime();

}

return totalSearchTime;

}

double Searcher::getTotalBuildTime() {

double totalBuildTime = 0;

for (TimeCounter timeCounter: timeCounters) {

totalBuildTime += timeCounter.getBuildTime();

}

return totalBuildTime;

}

void Searcher::addTimeCounter(TimeCounter newTimeCounter) {

timeCounters.push\_back(newTimeCounter);

}

TimeCounter Searcher::getLastTimeCounter() {

return timeCounters.back();

}

void Searcher::clear() {

arr.clear();

timeCounters.clear();

}

Searcher::Searcher() {

}

Searcher::Searcher(vector<int> arr) {

this->arr = arr;

}

Tiếp theo là prototype được định nghĩa cho NeedSortedSearcher.h

class NeedSortedSearcher: public Searcher {

protected:

bool isSorted;

vector<pair<int, int>> sortedIndexArr;

public:

template<typename T> bool isAscending(const vector<T> &arr) {

//Kiểm tra tăng

for (int i = 0; i < (int) arr.size() - 1; i++) {

if (arr[i] > arr[i+1]) {

return false;

}

}

return true;

}

void buildSortedIndexArr();

void clear() override;

NeedSortedSearcher();

NeedSortedSearcher(vector<int> arr);

};

Các triển khai của các hàm còn lại trong NeedSortedSearcher.cpp

void NeedSortedSearcher::buildSortedIndexArr() {

int size = (int) arr.size();

for (int i = 0; i < size; i++) {

sortedIndexArr.push\_back({arr[i], i});

}

if (isAscending(sortedIndexArr)) {

isSorted = true;

return;

}

sort(sortedIndexArr.begin(), sortedIndexArr.end());

isSorted = true;

}

void NeedSortedSearcher::clear() {

sortedIndexArr.clear();

Searcher::clear();

}

NeedSortedSearcher::NeedSortedSearcher(): Searcher() {

this->isSorted = false;

}

NeedSortedSearcher::NeedSortedSearcher(vector<int> arr): Searcher(arr) {

this->isSorted = false;

}

Cuối cùng là file BinarySearcher.h

class BinarySearcher: public NeedSortedSearcher {

public:

string getAlgorithm() override;

int search(int value) override;

BinarySearcher();

BinarySearcher(vector<int> arr);

};

Và file mã nguồn của nó để triển khai hàm search như sau:

string BinarySearcher::getAlgorithm() {

return "Binary Search";

}

int BinarySearcher::search(int value) {

TimeCounter newTimeCounter;

newTimeCounter.startBuild();

if(!isSorted) {

buildSortedIndexArr();

}

newTimeCounter.endBuild();

newTimeCounter.startSearch();

int left = 0, right = (int) sortedIndexArr.size() - 1;

while (left <= right) {

int mid = left + (right - left) / 2;

if (sortedIndexArr[mid].first == value) {

newTimeCounter.endSearch();

timeCounters.push\_back(newTimeCounter);

return sortedIndexArr[mid].second;

} else if (sortedIndexArr[mid].first < value) {

left = mid + 1;

} else {

right = mid - 1;

}

}

newTimeCounter.endSearch();

timeCounters.push\_back(newTimeCounter);

return -1;

}

BinarySearcher::BinarySearcher(): NeedSortedSearcher() {

}

BinarySearcher::BinarySearcher(vector<int> arr): NeedSortedSearcher(arr) {

}

Các thuật toán tìm kiếm khác cũng được triển khai tương tự như BinarySearcher.

3. Thiết kế giao diện

Giao diện được thiết kế bằng Qt, gồm một màn hình chính như hình dưới

Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, Phần mềm đa phương tiện

Mô tả được tạo tự động

Hình 4. Màn hình chính của ứng dụng SearchApp

Phần đầu tiên trên màn hình chính là phần để nhập dữ liệu vào, trong đó bao gồm 1 text area đầu tiên để chứa dữ liệu nhập từ bàn phím, một nút để nhập dữ liệu từ file, một box để quyết định có hiển thị dữ liệu nhập từ file hoặc bộ sinh lên giao diện không?. Tiếp đó là phần nhập size, loại sinh để bộ sinh dữ liệu tương ứng sinh dữ liệu đầu vào.

Field kế tiếp là để nhập các giá trị cần tìm kiếm.

Một list các lựa chọn tiếp theo là danh sách các thuật toán tìm kiếm mà người dùng chọn để tìm kiếm. Người dùng có thể chọn 1 hay nhiều thuật toán để thực hiện

Text area cuối cùng để in ra log tìm kiếm, có dạng sau

<<<Linear Search>>>

...Search 5321: find at 5320, build time = 0.000 ms, search time = 0.014 ms.

...Search 31231: find at 31230, build time = 0.000 ms, search time = 0.077 ms.

...Search 49999: find at 49998, build time = 0.000 ms, search time = 0.121 ms.

..Total: build time = 0.000 ms, search time = 0.212 ms.

Trong đó dòng đầu tiên là tên của thuật toán tìm kiếm được sử dụng. Mỗi dòng tiếp theo với dấu (…) ở đầu dùng để xuất ra kết quả tìm kiếm mỗi phần tử, in ra vị trí của nó trong dãy số, thời gian build (hay chuẩn bị các dữ liệu cần thiết, sắp xếp mảng, etc.) và thời gian search (tìm kiếm các phần tử trong dãy lí tưởng). Dòng cuối cùng, bắt đầu bằng dấu (..) chứa thông tin về tổng thời gian build và search tất cả các phần tử.

Sau đây là một ví dụ sử dụng giao diện này, với việc nhập dữ liệu từ bộ sinh tăng liên tiếp từ 1 tới 50000, nhập dãy này vào field, tìm kiếm 3 phần tử 5321, 31231, 49999 với 2 thuật toán tìm kiếm là Linear Search và Fibonacci Search.

Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, Phần mềm đa phương tiện

Mô tả được tạo tự động

Hình 5. Màn hình chạy thử với một thao tác mẫu

Có thể thấy Fibonacci Search có thời gian search nhanh hơn rất nhiều so với Linear Search (do độ phức tạp là O(log n) so với O(n)), nhưng bù lại, thời gian build một dãy lí tưởng để sử dụng Fibonacci Search là lâu hơn rất nhiều.

Cuối cùng, phần bên phải màn hình chứa các nút để vào giao diện chứa các animation để minh họa thuật toán tìm kiếm tương ứng với một dãy số được tạo sẵn. Trong màn hình này, mỗi phần tử của dãy sẽ được chứa trong một ô vuông xanh, các ô đang được so sánh với phần tử cần tìm kiếm sẽ được tô đỏ, ô được tìm thấy sẽ được tô xanh lá. Ngoài ra còn có thể có các label để chú thích về các chỉ số đặc biệt đang được so sánh, ví dụ như *left, right, mid, pos,* etc.

Ví dụ, khi ta ấn vào button Interpolation Search thì sẽ hiện đè màn hình sau

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ

Mô tả được tạo tự động

Hình 6. Màn hình animation minh họa cho Interpolation Search khi đang tìm kiếm

Ảnh có chứa ảnh chụp màn hình, văn bản, Phông chữ

Mô tả được tạo tự động

Hình 7. Màn hình animation minh họa cho Interpolation Search khi tìm thấy phần tử

Chương 4. Kết luận và hướng phát triển

1. Kết quả đạt được

Sau khi thực hiện đề tài, em đã hoàn thiện được một ứng dụng Search App thực hiện được các chức năng đã đề ra ở chương 1.

Về mã nguồn, các kết quả đạt được được tóm tắt trong bảng sau

|  |  |
| --- | --- |
| Tiêu chí |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |