**TRƯỜNG ĐẠI HỌC KỸ THUẬT CÔNG NGHIỆP**

**KHOA ĐIỆN TỬ**

**BỘ MÔN: CÔNG NGHỆ THÔNG TIN**

**A circular logo with a blue star and a yellow and red circle

Description automatically generated**

**TIỂU LUẬN**

**MÔN HỌC**

**CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT**

|  |  |  |
| --- | --- | --- |
| **Giảng viên hướng dẫn** | **:** | **Nguyễn Thị Hương** |
| **Họ tên sinh viên** | **:** | **La Đức Thắng** |
| **MSSV** | **:** | **K215480106120** |
| **Lớp** | **:** | **K58KMT.01** |

**THÁI NGUYÊN - 2024**

|  |  |
| --- | --- |
| **TRƯỜNGĐHKTCN** | **CỘNG HOÀ XÃ HỘI CHỦ NGHĨA VIỆT NAM** |
| **KHOA ĐIỆN TỬ** | **Độc lập - Tự do - Hạnh phúc** |

**TIỂU LUẬN**

**MÔN HỌC**: CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT.

BỘ MÔN : CÔNG NGHỆ THÔNG TIN

Sinh viên: La Đức Thắng

MSSV: K215480106120

Lớp: K58KMT.01 Ngành: Kỹ Thuật Máy Tính

Giáo viên hướng dẫn: Nguyễn Thị Hương

Ngày giao đề: 10/09/2024 Ngày hoàn thành: 11/10/2024

Tên đề tài :

Yêu cầu: Khảo sát, phân tích, thiết kế, cài đặt chương trình

|  |
| --- |
| **GIÁO VIÊN HƯỚNG DẪN** |
| (Ký và ghi rõ họ tên) |

**NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN**

Thái Nguyên, ngày 06 tháng 11 năm 2024

**GIÁO VIÊN HƯỚNG DẪN**

*(Ký ghi rõ họ tên)*

Mục lục

[LỜI NÓI ĐẦU 6](#_Toc181880328)

[LỜI CẢM ƠN 7](#_Toc181880329)

[CHƯƠNG I: ĐỆ QUY 8](#_Toc181880330)

[1.1. Khái niệm đệ quy 8](#_Toc181880331)

[1.2. Ví dụ 8](#_Toc181880332)

[1.2.1. Bài toán giai thừa 8](#_Toc181880333)

[1.2.2. Bài toán tháp Hà Nội 9](#_Toc181880334)

[1.2.3. Bài toán 8 quân hậu 10](#_Toc181880335)

[CHƯƠNG II: MẢNG VÀ DANH SÁCH 13](#_Toc181880336)

[2.1. Mảng 13](#_Toc181880337)

[2.2. Danh sách 14](#_Toc181880338)

[CHƯƠNG III: STACK VÀ QUEUE 16](#_Toc181880339)

[3.1. Stack 16](#_Toc181880340)

[3.1.1. Thêm phần tử 16](#_Toc181880341)

[3.1.2. Xóa phần tử 17](#_Toc181880342)

[3.2. Queue 19](#_Toc181880343)

[3.2.1. Thêm phần tử 19](#_Toc181880344)

[3.2.2. Xóa phần tử 19](#_Toc181880345)

[CHƯƠNG VI: DANH SÁCH MÓC NỐI 22](#_Toc181880346)

[4.1. Khái niệm 22](#_Toc181880347)

[4.2. Các loại danh sách móc nối 22](#_Toc181880348)

[4.2.1. Danh sách móc nối đơn 22](#_Toc181880349)

[4.2.2. Danh sách móc nối kép 26](#_Toc181880350)

[CHƯƠNG V: CÂY 30](#_Toc181880351)

[5.1. Các khái niệm 30](#_Toc181880352)

[5.1.1. Khái niệm cây 30](#_Toc181880353)

[5.1.2. Biểu diễn 30](#_Toc181880354)

# LỜI NÓI ĐẦU

Thuật toán và cấu trúc dữ liệu là nền tảng của khoa học máy tính và công nghệ phần mềm, cung cấp những kiến thức và công cụ cơ bản giúp sinh viên hiểu sâu hơn về cách thiết kế, phân tích và tối ưu hóa giải pháp cho các bài toán trong thực tế. Môn học Cấu trúc dữ liệu và Giải thuật (CTDLGT) trang bị cho chúng ta các kỹ thuật để tổ chức và xử lý dữ liệu một cách hiệu quả, từ đó giúp tối ưu hóa hiệu suất cho các ứng dụng và hệ thống.

Trong bài tiểu luận này, em xin trình bày một số các thuật toán cơ bản, phân tích hiệu suất của chúng, cùng với các cấu trúc dữ liệu phổ biến như mảng, danh sách liên kết, ngăn xếp, hàng đợi, cây, và đồ thị.

Mặc dù đã rất cố gắng để hoàn thành công việc, nhưng thời gian có hạn và thiếu kinh nghiệm cũng như kỹ năng chưa cao nên việc phân tích thiết kế còn nhiều thiếu sót, kính mong quý thầy cô và các bạn góp ý, bổ sung để em hoàn thiện bài tập tốt hơn nữa.

# LỜI CẢM ƠN

Trong suốt quá trình học tập và thực hiện bài tập lớn, em đã nhận được sự giúp đỡ tận tình của cô giáo Ths. Nguyễn Thị Hương thuộc bộ môn Tin học công nghiệp – Khoa Điện tử - Trường Đại học Kỹ thuật Công Nghiệp – Đại học Thái Nguyên. Em bày tỏ lòng biết ơn cô đã tận tình giúp đỡ, hướng dẫn em trong thời gian thực hiện đề tài này.

Mặc dù đã cố gắng hết sức, song do điều kiện thời gian và kinh nghiệm thực tế còn ít, cho nên đề tài không thể tránh khỏi thiếu sót. Vì vậy, em rất mong nhận được sự đóng góp ý kiến của các thầy giáo, cô giáo và các bạn.

Em xin chân thành cảm ơn!

Sinh viên thực hiện

La Đức Thắng

# CHƯƠNG I: ĐỆ QUY

## Khái niệm đệ quy

Đệ quy (tiếng Anh: recursion) là phương pháp dùng trong các chương trình máy tính trong đó có một hàm tự gọi chính nó.

## Ví dụ

### Bài toán giai thừa

Trong toán học, giai thừa (Tiếng Anh: factorial) là một toán tử một ngôi trên tập hợp các số tự nhiên. Cho n là một số tự nhiên dương, "n giai thừa", ký hiệu n!

\*Trường hợp n=0:

Giải thuật:

int giai\_thua(int n)

{

if (n == 0) return 1;

else return n \* giai\_thua(n - 1);

}

Kết quả:

A screenshot of a computer

Description automatically generated

### Bài toán tháp Hà Nội

Tháp Hà Nội là một trò chơi toán học kinh điển thường được sử dụng để giải các bài toán về đệ quy.

Luật chơi:

* Chỉ có 3 cột để di chuyển.
* Một lần chỉ được di chuyển một đĩa (không được di chuyển đĩa nằm giữa).
* Một đĩa chỉ có thể được đặt lên một đĩa lớn hơn (không nhất thiết hai đĩa này phải có kích thước liền kề, tức là đĩa nhỏ nhất có thể nằm trên đĩa lớn nhất).

Giải thuật:

void ThapHaNoi(int n, char a, char b, char c) {

if (n == 1) {

cout << "\t" << a << " chuyen sang " << c << endl; ///

return;

}

ThapHaNoi(n - 1, a, c, b);

ThapHaNoi(1, a, b, c);

ThapHaNoi(n - 1, b, a, c);

}

* n là số đĩa cần di chuyển
* a là cột nguồn
* b là cột trung gian
* c là cột đích

Chuyển n-1 đĩa từ a sang b, c làm cột trung gian.

Chuyển đĩa lớn nhất từ a sang c, b làm cột trung gian.

Chuyển n-1 đĩa từ b sang c, a làm cột trung gian.

Kết quả:

A screenshot of a computer screen

Description automatically generated

### Bài toán 8 quân hậu

Bài toán tám quân hậu là bài toán đặt tám quân hậu trên bàn cờ vua kích thước 8×8 sao cho không có quân hậu nào có thể ăn được quân hậu khác, hay nói khác đi không quân hậu nào có để di chuyển theo quy tắc cờ vua. Màu của các quân hậu không có ý nghĩa trong bài toán này. Như vậy, lời giải của bài toán là một cách xếp tám quân hậu trên bàn cờ sao cho không có hai quân nào đứng trên cùng hàng, hoặc cùng cột hoặc cùng đường chéo. Bài toán tám quân hậu có thể tổng quát hóa thành bài toán đặt n quân hậu trên bàn cờ n×n(n ≥ 4).

Giải thuật:

Đàu tiên ta khởi tạo một ma trận 8x8 để mô phỏng bàn cờ:

vector<vector<int> > board(8, vector<int>(8, 0));

Tiếp đến kiểm tra xem vị trí (row, col) có thể đặt được quân hậu hay không

bool Ok(int row, int col) {

**//Vòng lặp kiểm tra cột**

for (int i = 0; i < row; i++)

{

if (board[i][col] == 1) return false;

}

**//Vòng lặp kiểm tra đường chéo trái**

for (int i = row, j = col; i >= 0 && j >= 0; i--, j--)

{

if (board[i][j] == 1) return false;

}

**//Vòng lặp kiểm tra đường chéo phải**

for (int i = row, j = col; i >= 0 && j < 8; i--, j++)

{

if (board[i][j] == 1) return false;

}

return true;

}

Tiếp đến, ta có hàm đệ quy Try(int row)

bool Try(int row) {

if (row >= 8) return true;

for (int col = 0; col < 8; col++) {

if (Ok(row, col)) {

board[row][col] = 1;

if (Try(row + 1)) return true;

board[row][col] = 0;

}

}

return false;

}

Hàm đệ quy này tìm cách đặt quân hậu trên bảng. Nếu đã đặt hết 8 quân hậu thì trả về true. Nếu không, thử đặt quân hậu vào từng cột trong hàng hiện tại. Nếu không thể đặt quân hậu nào trong hàng này mà không vi phạm, quay lui bằng cách bỏ quân hậu đã đặt ở hàng trước.

Hàm in ra kết quả:

void Xuat() {

for (int i = 0; i < 8; i++) {

for (int j = 0; j < 8; j++)

cout << (board[i][j] ? "Q " : ". ");

cout << endl;

}

}

Kết quả:

A pattern of white letters on a black background

Description automatically generated

# CHƯƠNG II: MẢNG VÀ DANH SÁCH

## 2.1. Mảng

Mảng (array) là một tập hợp có thứ tự gồm một số cố định, các phần tử không có phép bổ xung hoặc loại bỏ1 phần tử. Chỉ có các phép tạolập, tìm kiếm, lưu trữ. Mỗi phần tử của mảng ngoài giá trị (info) còn được đặc trưng bởi chỉ số để biểu hiện thứ tự của nó trong mảng. Có mảng 1 chiều, 2, 3 chiều...n chiều.

Khai báo mảng: <kiểu dữ liệu> <tên mảng>[kích cỡ mảng]

Ví dụ: int a[100];

***Cấu trúc lưu trữ của mảng***

Một véc tơ A có n phần tử, nếu mỗi phần tử a[i] (1<=i<=n) chiếm c từ máy thì nó sẽ được lưu trữ trong cn từ máy kế tiếp nhau (lưu trữ kế tiếp – Sequential storange allocation)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | … | … |  | … | … | … |  |

|  |
| --- |
| L |

L0: địa chỉ gốc (là địa chỉ của từ máy đầu tiên trong miền nhớ kế tiếp dùng để lưu trữ véc tơ )

Địa chỉ được tính bởi công thức:

* là địa chỉ cơ sở của mảng (địa chỉ của phần tử đầu tiên ​
* là chỉ số của phần tử (với iii bắt đầu từ 1 nếu theo kiểu lập chỉ mục 1-based, hoặc từ 0 nếu 0-based),
* là kích thước của mỗi phần tử trong mảng

gọi là hàm địa chỉ (address function)

Ví dụ: Ma trận 2x3 với và được lưu trữ theo cột như sau:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |

Vị trí của phần tử :

## 2.2. Danh sách

Danh sách Là một tập có thứ tự nhưng bao gồm một số biến động các phần tử. Phép bổ sung và loại bỏ một phần tử là phép thường xuyên tác động lên danh sách. Một danh sách mà quan hệ lân cận giữa các phần tử được hiển thị ra gọi là danh sách tuyến tính.

Danh sách tuyến tính là một dãy gồm các phẩn tử x1, x2, ...., xn có cùng một kiểu dữ liệu trong đó: n được gọi là số lượng phần tử của danh sách, nếu n=0 thì gọi là danh sách rỗng.

Các phép toán trên danh sách

* Bổ sung vào danh sách một phần tử mới trước hoặc sau một phần tử nào đó thuộc danh sách
* Xóa một phần tử khỏi danh sách
* Truy cập đến một phần tử nào đó thuộc danh sách
* Tìm số lượng phần tử trong danh sách
* Duyệt (đọc) danh sách từ trái sang phải hoăc từ phải sang trái

Ngoài ra còn có các phép toán:

* Ghép 2 hoặc nhiều danh sách
* Tách một danh sách thành nhiều danh sách
* Sao chép danh sách
* Cập nhật danh sách
* Sắp xếp các phần tử trong danh sách theo một thứ tự nhất định

Khai báo danh sách: list<kiểu dữ liệu> tên list;

Ví dụ: list<int> numbers;

//Thêm phần tử vào danh sách:

numbers.push\_back(10);

numbers.push\_front(5);

Chú ý: Cần khai báo thư viện <list> để sử dụng danh sách.

# CHƯƠNG III: STACK VÀ QUEUE

## 3.1. Stack

Stack (ngăn xếp) là một cấu trúc dữ liệu tuân theo nguyên tắc LIFO (Last In, First Out), nghĩa là phần tử được thêm vào sau cùng sẽ là phần tử được lấy ra đầu tiên. Trong chương này, em sẽ sử dụng mảng để mô phỏng cả stack và queue.

Hàm displayArray dùng để hiển thị mảng:

void displayArr(int a[], int i, int size) {

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

if (i == 0) {

cout << "{" << a[i] << "; ";

}

else if (i == size - 1) {

cout << a[i] << "}";

}

else {

cout << a[i] << "; ";

}

}

}

### 3.1.1. Thêm phần tử

Khi stack đầy, không thêm vào stack nữa. Nếu stack chưa đầy, phần tử trên cùng là phần tử vừa được thêm vào stack.

void push(int x, int &top, int size, int a[]) {

if (top >= size - 1) {

cout << "Stack day\n";

}

else {

top++;

a[top] = x;

}

}

### 3.1.2. Xóa phần tử

Để xóa một phần tử, đầu tiên ta lấy phần tử a[top] ra khỏi stack, từ đó, phần tử a[top-1] sẽ ở vị trí top. Nếu stack rỗng, không thể xóa.

void pop(int &top) {

if (top <= -1) {

cout << "Stack rong\n";

}

else {

top--;

}

}

};

Kết hợp 2 thuật toán, thực hiện thêm phần tử vào stack:

int main() {

int size=0, i;

Stack s;

cout<<"Nhap kich co cua stack: ";cin>>size;

int top = size - 1;

int a[size];

int rear=size-1;

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

cout<<"Nhap a["<<i<<"]: ";

cin>>a[i];

}

displayArr(a, i, size);

int x;

cout << "\nNhap phan tu muon them vao stack: "; cin >> x;

cout << "Lay phan tu a[" << size - 1 << "] ra khoi stack" << endl;

s.pop(top);

cout << "Them phan tu moi vao vi tri a[" << size - 1 << "]" << endl;

s.push(x, top, size, a);

displayArr(a, i, size);

}

Kết quả:

A computer screen shot of a black screen

Description automatically generated

## 3.2. Queue

Queue (hàng đợi), ngược lại với stack. là một cấu trúc dữ liệu tuân theo nguyên tắc FIFO (First In, First Out), nghĩa là phần tử được thêm vào sau đầu tiên sẽ là phần tử được lấy ra đầu tiên. Queue còn có thể được gọi là “Stack kiểu FIFO”.

### 3.2.1. Thêm phần tử

void queue\_them(int queue[], int x, int rear, int size)

{

if (rear == size) {

cout << "Queue day, khong the them!";

}

else {

queue[rear] = x;

}

}

Ta có mảng queue[] là nơi lưu trữ các phần tử của queue, biến x là phần tử cần thêm vào queue, rear là phần tử cuối cùng trong queue, size là kích cỡ của queue.

Nếu rear=size, khi đó queue đã đầy, còn không thì x sẽ được gán cho giá trị queue[rear]

### 3.2.2. Xóa phần tử

void queue\_xoa(int queue[], int x, int front, int rear)

{

if (rear == 0) {

cout << "Khong con phan tu nao trong queue";

}

else {

int j = 0;

while (j < rear) {

queue[j] = queue[j + 1];

j++;

}

}

}

Xóa phần tử sẽ có biến front cho biết vị trí đầu tiên của hàng đợi. Hàm này sẽ xóa phần tử đầu tiên được thêm vào queue.

Kết hợp 2 thuật toán, thực hiện chức năng thêm phần tử vào queue:

int main() {

int size=0, i;

Queue q;

cout<<"Nhap kich co cua queue: ";cin>>size;

int top = size - 1;

int a[size];

int rear=size-1;

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

cout<<"Nhap a["<<i<<"]: ";

cin>>a[i]; }

displayArr(a, i, size);

int x;

cout << "\nNhap phan tu muon them vao queue: "; cin >> x;

q.queue\_xoa(a, x, 0, size);

cout << "Xoa phan tu a[0]: " << a[0];

cout << "\n";

displayArr(a, i, size-1);

cout << "\nThem phan tu moi vao vi tri a[" << size-1<<"] ";

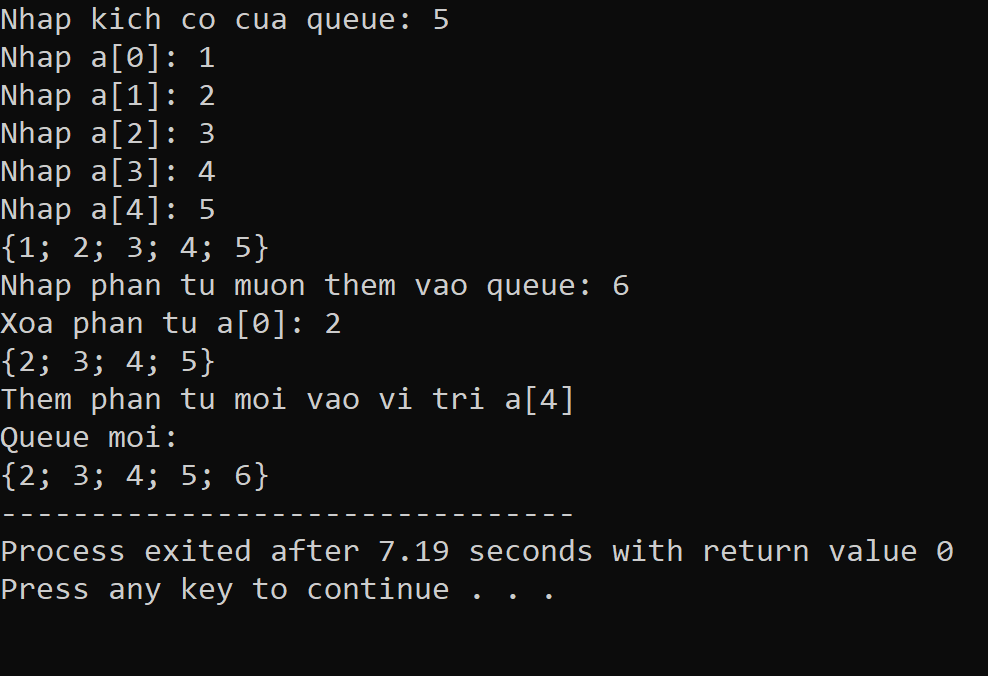
q.queue\_them(a, x, size - 1, size);

cout << "\nQueue moi: "<<endl;

displayArr(a, i, size);

}

Kết quả:



# CHƯƠNG VI: DANH SÁCH MÓC NỐI

## 4.1. Khái niệm

Danh sách móc nối (linked list) là một cấu trúc dữ liệu động trong lập trình, cho phép lưu trữ một tập hợp các phần tử, trong đó mỗi phần tử (được gọi là nút - node) chứa một giá trị và một hoặc nhiều con trỏ (hoặc tham chiếu) trỏ đến các nút khác trong danh sách.

Ưu điểm:

* Cung cấp giải pháp để chứa cấu trúc dữ liệu tuyến tính.
* Dễ dàng thêm hoặc xóa các phần tử trong danh sách mà không cần phải cấp phát lại.
* Cấp phát bộ nhớ động.

Nhược điểm: Một số danh sách móc nối đơn giản không cho phép truy cập ngẫu nhiên dữ liệu.

## 4.2. Các loại danh sách móc nối

### 4.2.1. Danh sách móc nối đơn

Là danh sách móc nối có nhiều nút móc nối đơn lại với nhau. Nút bao gồm:

* Một số từ máy kế tiếp nhau
* Nút có thể nằm ở vị trí bất kỳ trong bộ nhớ
* Mỗi nút chứa thông tin ứng với 1 phần tử và địa chỉ phần tử đứng sau

Khai báo nút:

struct Node {

int data; //Dữ liệu của nút hiện tại

Node\* next; //Nút sau

Node\* prev; //Nút trước

Node(int val) : data(val), next(nullptr), prev(nullptr) {}

};

Khai báo danh sách:

class DS\_Don {

private:

Node\* head;

public:

DS\_Don() : head(NULL) {}

Node\* current = head;

while (current) {

cout << current->data << " -> ";

current = current->next;

}

cout << "NULL" << endl;

}

~DS\_Don() {

Node\* current = head;

while (current) {

Node\* next = current->next;

delete current;

current = next;

}

}

};

Một số phép toán:

* Thêm nút:
  + Tạo một nút mới với dữ liệu muốn thêm. Con trỏ của nút này (next) sẽ được đặt là NULL hoặc giá trị mặc định.
  + Nếu danh sách hiện tại trống (nghĩa là head đang trỏ đến NULL), nút mới sẽ trở thành nút đầu tiên và head sẽ trỏ đến nút này.
  + Nếu danh sách không trống, cần duyệt qua các nút từ head cho đến khi tìm thấy nút cuối cùng (nút có next == NULL).
  + Đặt con trỏ next của nút cuối cùng (đã tìm được ở bước 3) trỏ đến nút mới, do đó nút mới sẽ trở thành nút cuối trong danh sách.

void append(int val) {

Node\* newNode = new Node(val);

if (!head) {

head = newNode;

} else {

Node\* temp = head;

while (temp->next) {

temp = temp->next;

}

temp->next = newNode;

}

}

A screenshot of a computer program

Description automatically generated

* Xóa nút có giá trị ngẫu nhiên:
  + Kiểm tra danh sách có rỗng không
  + Kiểm tra dữ liệu của nút muốn xóa có tồn tại trong danh sách không
  + Duyệt cả danh sách để tìm nút chứa giá trị cần xóa.

void deleteValue(int x) {

if (!head) return;

if (head->data == x) {

Node\* temp = head;

head = head->next;

delete temp;

return;

}

Node\* current = head;

while (current->next && current->next->data != x) {

current = current->next;

}

if (current->next) {

Node\* temp = current->next;

current->next = current->next->next;

delete temp;

}

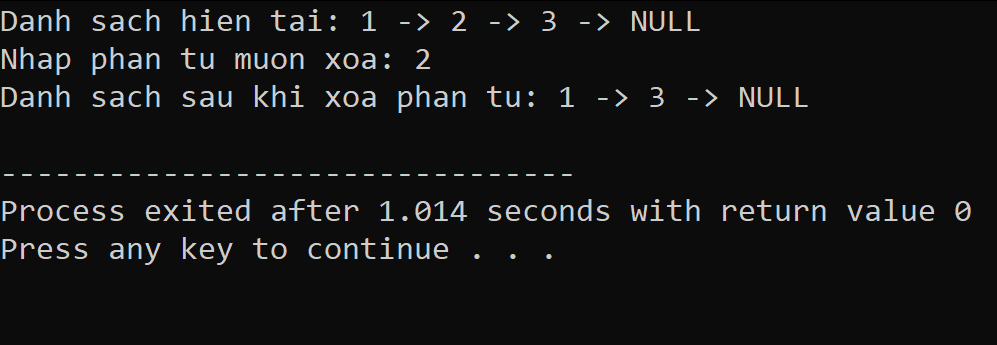
else {

cout << "Nut khong ton tai" << endl;

}

}

Kết quả:



### 4.2.2. Danh sách móc nối kép

Là dạng danh sách nơi mỗi phần tử sẽ liên kết với phần tử đứng trước và sau nó trong danh sách. Mỗi phần tử gồm biến dữ liệu và 2 con trỏ liên kết tới phần tử trước và sau nó.

Khai báo danh sách:

class DS\_Kep {

private:

Node\* head;

public:

DS\_Kep() : head(nullptr) {}

void add\_Dau(int value) {

Node\* newNode = new Node(value);

if (!head) {

head = newNode;

}

else {

newNode->next = head;

head->prev = newNode;

head = newNode;

}

}

~DS\_Kep() {

while (head) {

deleteHead();

}

}

};

Một số phép toán:

* Thêm một nút vào đầu danh sách
  + Tạo một nút mới
  + Kiểm tra danh sách có rỗng không. Nếu rỗng, nút mới là nút đầu tiên
  + Nếu danh sách không rỗng, nút mới sẽ trở thành nút đứng trước của nút đầu.

void add\_Dau(int value) {

Node\* newNode = new Node(value);

if (!head) {

head = newNode;

}

else {

newNode->next = head;

head->prev = newNode;

head = newNode;

}

}

A screenshot of a computer

Description automatically generated

* Thêm một nút vào cuối danh sách: Tương tự như thêm phần tử vào đầu, nhưng lần này nút mới sẽ là nút đứng sau của nút cuối.

void add\_Cuoi(int value) {

Node\* newNode = new Node(value);

if (!head) {

head = newNode;

}

else {

Node\* current = head;

while (current->next) {

current = current->next;

}

current->next = newNode;

newNode->prev = current;

}

}

A screenshot of a computer

Description automatically generated

* Xóa một nút khỏi danh sách:
  + Duyệt danh sách để tìm phần tử cần xóa.
  + Nếu nút cần xóa là nút đầu, con trỏ head sẽ trỏ đến nút kế tiếp trong danh sách.
  + Nếu nút cần xóa là nút giữa, cập nhật 2 nút prev của nút sau và next của nút trước
  + Nếu nút cần xóa là nút cuối, cập nhật nút tiếp theo của nút đứng trước nó.

void deleteNode(int value) {

Node\* current = head;

while (current) {

if (current->data == value) {

if (current->prev) {

current->prev->next = current->next;

}

else {

head = current->next;

}

if (current->next) {

current->next->prev = current->prev;

}

delete current;

return;

}

current = current->next;

}

cout << "Nut khong ton tai" << endl;

}

A screenshot of a computer

Description automatically generated

# CHƯƠNG V: CÂY

## 5.1. Các khái niệm

### 5.1.1. Khái niệm cây

Cây là tập hợp các nút sao cho có một nút được gọi là một nút gốc, các nút còn lại được phân thành n tập riêng biệt T1, T2,…,Tn. Mỗi tập Ti là 1 cây. Giữa các nút có mối quan hệ phân cấp gọi là cha-con.

Cây không có nút được gọi là cây rỗng (null tree).

### 5.1.2. Biểu diễn

A diagram of a network

Description automatically generated

### 5.1.3. Thuật ngữ

**Bậc của nút và bậc của cây:**

* Bậc của nút là số cây con của nó
* Bậc của cây là bậc lớn nhất của các nút trong cây

**Nút gốc, nút lá, nút nhánh:**

* Nút gốc là nút không có nút cha
* Nút lá là nút có bậc là 0
* Nút nhánh là nút không là lá mà cũng không là gốc

**Đường đi:** là một chuỗi các cạnh nối liên tiếp giữa hai nút trong cây. Đường đi bắt đầu từ một nút nguồn và kết thúc tại một nút đích , có thể từ nút gốc (root) đến một nút lá hoặc giữa bất kỳ hai nút nào trong cây.

**Mức của nút và chiều cao của cây:**

* Mức của nút gốc = 1, mức của nút con = mức của nút cha + 1
* Chiều cao của cây là mức lớn nhất của các nút lá

**Nút tổ tiên của một nút:** Là các nút nằm trên đường đi từ cuối lên đầu

**Cây có thứ tự:** Một cây con gọi là có thứ tự khi ta thay vị trí của các cây con và nhận được cây mới.

**Rừng:** Là tập hợp hữu hạn các cây phân biệt. Nếu bỏ đi nút gốc của một cây, ta được một rừng.

## 5.2. Cây nhị phân

### 5.2.1. Các khái niệm

Cây nhị phân: Là cây mà một nút cha chỉ có nhiều nhất 2 nút con

**A diagram of a network

Description automatically generated**

Tính chất: Số nút tối đa của mức i trong cây là

Số nút tối đa trong cây: (n=chiều cao)

Chiều cao: (N=số nút)

Cây nhị phân hoàn chỉnh: Các nút ở các mức đều đạt tối đa trừ mức cuối

Cây nhị phân đầy đủ: Các nút đạt tối đa ở mọi mức

Cây nhị phân gần đầy: Các nút trừ nút cuối ở các mức đều đạt tối đa nhưng không dạt đều về bên trái

### 5.2.1. Tổ chức lưu trữ

**Sử dụng mảng 1 chiều**: Đánh số thứ tự từ gốc. Tại mỗi mức, đánh số các nút từ trái sang phải, từ mức thấp đến mức cao.

* Con của nút thứ i là và
* Con của nút thứ j là

Tuy nhiên, cách lưu trữ này có nhược điểm là gây lãng phí bộ nhớ.

**Lưu trữ liên kết:**

* Quản lý cây thông qua nút gốc.
* Mỗi nút gồm có dữ liệu và 2 con trỏ pLeft, pRight liên kết đến nút con trái và nút con phải của nó.
* Nút lá có 2 liên kết trái phải đều rỗng.

Cấu trúc:

class Bin\_Tree {

public:

int data;

Bin\_Tree\* pLeft;

Bin\_Tree\* pRight;

Bin\_Tree(int val) {

data = val;

pLeft = nullptr;

pRight = nullptr;

}

};

Bin\_Tree\* root = nullptr;

**Thêm một nút vào cây:**

* Nếu nút gốc rỗng, nút mới được thêm vào sẽ là nút gốc.
* Nếu giá trị cần thêm nhỏ hơn giá trị của nút gốc, thêm vào cây con trái
* Nếu giá trị cần thêm lớn hơn giá trị của nút gốc, thêm vào cây con phải

Bin\_Tree\* insert(int value, Bin\_Tree\* root) {

if (root == nullptr) {

return new Bin\_Tree(value);

}

if (value < root->data) {

root->pLeft = insert(value, root->pLeft);

}

else {

root->pRight = insert(value, root->pRight);

}

return root;

}

**Tìm kiếm một nút có giá trị ngẫu nhiên trong cây:**

* Duyệt qua toàn bộ cây và lưu trữ tất cả các nút vào một danh sách.
* Sau khi có danh sách chứa các nút, sử dụng một hàm ngẫu nhiên để chọn một phần tử từ danh sách.

Bin\_Tree\* find(int val, Bin\_Tree\* root) {

if (root == nullptr) {

return nullptr;

}

if (val < root->data) {

return find(val, root->pLeft);

}

else if (val > root->data) {

return find(val, root->pRight);

}

else {

return root;

}

}

A screenshot of a computer program

Description automatically generated

**Xóa một nút khỏi cây:**

* Nếu cây rỗng, trả về nullptr.
* Trường hợp đã tìm thấy nút cần xóa:
* Nếu nút cần xóa không có con: Xóa nút
* Nếu nút cần xóa có 1 con:
  + Chọn con duy nhất
  + Xóa nút hiện tại
  + Trả về con của nút này để nối vào cây
* Nếu nút cần xóa có 2 con:
  + Chọn cây con phải
  + Tìm nút nhỏ nhất trong cây con phải
  + Thay thế giá trị của nút cần xóa bằng giá trị của nút thay thế
  + Xóa nút thay thế trong cây con phải

Bin\_Tree\* del(int val, Bin\_Tree\* root) {

if (root == nullptr) return nullptr;

if (val < root->data) {

root->pLeft = del(val, root->pLeft);

}

else if (val > root->data) {

root->pRight = del(val, root->pRight);

}

else {

if (!root->pLeft && !root->pRight) {

delete root;

return nullptr;

}

else if (!root->pLeft || !root->pRight) {

Bin\_Tree\* child = (root->pLeft) ? root->pLeft : root->pRight;

delete root;

return child;

}

else {

Bin\_Tree\* successor = root->pRight;

while (successor->pLeft) {

successor = successor->pLeft;

}

root->data = successor->data;

root->pRight = del(successor->data, root->pRight);

}

}

return root;

}

A screen shot of a computer

Description automatically generated

### 5.2.3. Duyệt cây nhị phân

Phép duyệt là phép xử lý các nút trên cây, mỗi nút một lần.

Ví dụ, ta có cây sau:

A diagram of a network

Description automatically generated

Duyệt theo thứ tự trước:

* Thăm gốc.
* Duyệt cây con trái theo thứ tự trước.
* Duyệt cây con phải theo thứ tự trước.

void duyetTruoc(Bin\_Tree\* node) {

if (node != nullptr) {

cout << node->data << " ";

duyetTruoc(node->pLeft);

duyetTruoc(node->pRight);

}

}

Duyệt theo thứ tự giữa:

* Duyệt cây con trái theo thứ tự giữa
* Thăm gốc
* Duyệt cây con phải theo thứ tự giữa

void duyetGiua(Bin\_Tree\* node) {

if (node != nullptr) {

duyetGiua(node->pLeft);

cout << node->data << " ";

duyetGiua(node->pRight);

}

}

Duyệt theo thứ tự sau:

* Duyệt cây con trái theo thứ tự sau
* Duyệt cây con phải theo thứ tự sau
* Thăm gốc

void duyetSau(Bin\_Tree\* node) {

if (node != nullptr) {

duyetSau(node->pLeft);

duyetSau(node->pRight);

cout << node->data << " ";

}

}

A computer screen shot of numbers and symbols

Description automatically generated

# CHƯƠNG VI: ĐỒ THỊ

## 6.1. Các khái niệm

Đồ thị G=(V, E) là một cấu trúc dữ liệu biểu diễn mối quan hệ giữa các đối tượng

* V là tập hợp hữu hạn các phần tử
* E là tập hợp hữu hạn các cạnh cung

Nếu (v1, v2) là cặp đỉnh thuộc E thì ta nói: có một cung nối v1 và v2.

* Nếu cung (v1, v2) khác với cung (v2, v1) thì ta có một đô thị định hướng (directed graph hay digraph). Lúc đó (v1, v2) được gọi là cung định hướng v1, v2.

A diagram of a network

Description automatically generated

Đồ thị có hướng

* Nếu thứ tự các nút trên cung không được coi trọng thì ta gọi đồ thị không định hướng (undirected graph)

A diagram of a network

Description automatically generated

Đồ thị vô hướng

Một số khái niệm:

* Một đường đi (path) từ đỉnh v, đến đỉnh vị trong đồ thị G là một dãy đỉnh mà là các cung trong E(G). Số lượng các cung trên đường đi ấy gọi là độ dài của đường đi (path length).
* Một đường đi đơn (simple path) là đường đi mà mọi đỉnh trên đó, trừ đỉnh đầu và đỉnh cuối, đều khác nhau.
* Một chu trình (cycle) là một đường đi đơn mà đỉnh đầu và đỉnh cuối trùng nhau.

## 6.2. Biểu diễn đồ thị

**Đồ thị biểu diễn bằng ma trận kề liên kết:** Xét một đồ thị G(V, E) định hướng với V gồm có n đỉnh (n≥1) mà giả sử các đỉnh đã được đánh số theo một quy định nào đó. Ma trận lân cận A biểu diễn G là một ma trận vuông kích thước nxn. Các phần tử của ma trận có giá trị 0 hoặc 1.

* Nếu phần tử aj = 1, tồn tại một cung định hướng (Vi, Vj) trong E.
* Không tồn tại cung aj = 0

**Đồ thị biểu diễn bằng danh sách kề:** Danh sách kề là 1 mảng các danh sách. Ở đây, n hàng của ma trận kề thay bằng n danh sách liên kết động. Mỗi đỉnh của G có 1 danh sách, mỗi nút trong danh sách thể hiện các đỉnh lân cận nút này

struct Canh {

int u, v, w; // Cạnh từ u đến v với trọng số w

};

class Graph {

private:

int dinh;

vector<vector<pair<int, int>>> adjList;

vector<Canh> canh;

public:

Graph(int vertices) : dinh(vertices), adjList(vertices) {}

}

## 6.3. Các thuật toán

### 6.3.1. Thuật toán duyệt đồ thị

Có 2 cách duyệt đồ thị là duyệt theo chiều sâu (DFS) và duyệt theo chiều rộng (BFS).

**Duyệt theo chiều sâu:** Xuất phát từ đỉnh V, xét đỉnh A là đỉnh lân cận đỉnh V chưa được thăm. Nếu 1 đỉnh V đã được thăm mà mọi điểm của nó đều vừa được thăm thì ta sẽ quay ngược lên đỉnh cuối cùng vừa được thăm mà có đỉnh w chưa được thăm và lặp lại cho tới khi không một nút nào không được thăm.

Procedure DFS(v)

{

Visited(v) = 1;

for (mỗi đỉnh w lân cận của v)

do {

if (Visited(w) == 0) DFS(w);

}

}

**Duyệt theo chiều rộng:** Xuất phát từ đỉnh V, lần lượt thăm tất cả các lân cận chưa được thăm của V. Sau đó tiếp tục lần lượt thăm các lân cận của các lân cận của V chưa được thăm.

Procedure BFS(v)

{

Visited(v) = 1;

Khởi tạo q và v đã được nạp vào;

while (q không rỗng) {

call(qDelete(v, q));

}

for (mỗi đỉnh w lân cận của v)

{

if (Visited(w) == 0) {

call(qInsert(w, q));

}

}

} return;

### 6.3.2. Thuật toán Kruskal

Thuật toán Kruskal được dùng để tìm cây khung nhỏ nhất.

Cây khung là tập hợp bao gồm các khung thuộc một phép duyệt từ 1 đỉnh đến các đỉnh còn lại trong V. Một đồ thị có thể có nhiều cây khung.

Giải thuật Kruskal:

* Sắp xếp các cung theo thứ tự tối giản đối với trọng số
* Bắt đầu từ T = , lặp đến khi không còn đỉnh nào.
* Lấy ra cung w có trọng số nhỏ nhất và thêm vào T với điều kiện không tạo chu trình.

// Hàm tìm kiếm và hợp nhất cho thuật toán Kruskal

int find(vector<int>& parent, int i) {

if (parent[i] == -1)

return i;

return find(parent, parent[i]);

}

void unionSet(vector<int>& parent, int x, int y) {

parent[x] = y;

}

// Thuật toán Kruskal

void kruskal() {

sort(canh.begin(), canh.end(), [](Canh a, Canh b) {

return a.w < b.w;

});

vector<int> parent(dinh, -1);

cout << "\nCay khung toi thieu (Kruskal):" << endl;

for (auto& edge : canh) {

int x = find(parent, edge.u);

int y = find(parent, edge.v);

if (x != y) {

cout << edge.u << " - " << edge.v << " : " << edge.w << endl;

unionSet(parent, x, y);

}

}

}

Ví dụ với đồ thị Ga có giá trị như sau:

A diagram of a network

Description automatically generated

Cây khung tối thiểu sẽ là:

A diagram of a diagram

Description automatically generated

A black screen with a black background

Description automatically generated

### 6.3.3. Thuật toán Dijkstra

* Còn gọi là thuật toán tìm đường đi ngắn nhất.
* Xét đồ thị có hướng G=(V, E) với |V|=n
* Ma trận trọng số
* Ta có là điểm xuất phát, là chiều dài nhỏ nhất từ s đến
* Bắt đầu duyệt từ đỉnh s, gán giá trị cho nếu . Nếu không, . Lặp lại cho đến khi duyệt hết các đỉnh.
  + Chọn đỉnh w chưa duyệt có nhỏ nhất, duyệt đỉnh này
  + Với các đỉnh chưa duyệt khác:

void dijkstra(int start) {

vector<int> dist(dinh, INT\_MAX);

priority\_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;

pq.push({ 0, start });

dist[start] = 0;

while (!pq.empty()) {

int u = pq.top().second;

pq.pop();

for (auto& p : adjList[u]) {

int v = p.first;

int weight = p.second;

if (dist[u] + weight < dist[v]) {

dist[v] = dist[u] + weight;

pq.push({ dist[v], v });

}

}

}

cout << "Duong di ngan nhat tu dinh " << start << " den cac dinh khac:" << endl;

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

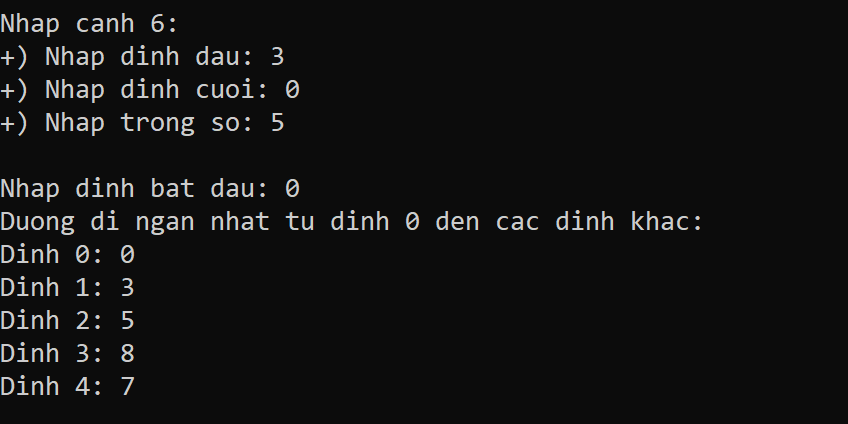
cout << "Dinh " << i << ": "

<< (dist[i] == INT\_MAX ? "Khong co duong di" : toString(dist[i])) << endl;

}

}

Tiếp tục xét với đồ thị Ga đã nêu ở trên, tìm đường đi ngắn nhất từ đỉnh 0 đến các đỉnh khác:



# CHƯƠNG VII: SẮP XẾP

Sắp xếp là quá trình tổ chức lại tập dữ liệu theo 1 trật tự tăng hoặc giảm dần. Trong chương này, em sẽ sử dụng các thuật toán để sắp xếp các phần tử theo thứ tự tăng dần.

## 7.1. Sắp xếp chọn lọc (Selection Sort)

Giải thuật sắp xếp một danh sách các giá trị bằng cách lặp lại việc đặt 1 giá trị cụ thể vào đúng vị trí thích hợp của nó trong dãy sắp xếp. Với mỗi vị trí trong danh sách, giải thuật tìm giá trị phù hợp cho vị trí đó

void SelectionSort(int a[], int n) {

for (int i = 0; i < n - 1; i++) {

int min = i;

}

for (int j = i + 1; j < n; j++) {

if (a[j] < a[min]) {

min = j;

}

}

if (min != i) {

swap(a[i], a[min]);

}

}

}

## 7.2. Sắp xếp chèn (Insertion Sort)

Ở bước thứ i, các phần tử được sắp xếp theo thứ thự khóa . Xét phần tử thứ i, có khóa , lần lượt so sánh các phần tử đã được sắp sẵn để tìm vị trí chèn thích hợp

Ví dụ: Sắp xếp các phần tử theo thứ tự tăng dần

void InsertionSort(int a[], int n)

{

int holePosition;

int valueToInsert;

for (int i = 0; i < n; i++)

{

valueToInsert = a[i];

holePosition = i;

while (holePosition > 0 && a[holePosition - 1] > valueToInsert)

{

a[holePosition] = a[holePosition - 1];

holePosition = holePosition - 1;

}

a[holePosition] = valueToInsert;

}

}

## 7.3. Sắp xếp nổi bọt (Bubble Sort)

Ở bước thứ i, kể từ phần tử thứ i, so sánh 2 phần tử kề nhau. Nếu khóa của phần tử trước lớn hơn khóa của phần tử sau thì đổi chỗ.

void BubbleSort(int a[], int n) {

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - 1 - i; j++) {

if (a[j] > a[j + 1]) {

swap(a[j], a[j + 1]);

}

}

}

}

## 7.4. Sắp xếp nhanh (Quick Sort)

(1) Xét một dãy n phần tử. Chọn x = a[(n+1) div 2] làm khóa. Đi từ 2 đầu của dãy, nếu một cặp thì hoán vị 2 phần tử này. Tăng , giảm . Lặp lại cho đến khi i>j.

(2) Lặp lại (1) với cả 2 phân đoạn A và B. Kết thúc khi các phân đoạn thu được có length = 1

int partitionFunc(int left, int right, int pivot)

{

int leftPointer = left - 1;

int rightPointer = right;

while (1)

{

while (a[++leftPointer] < pivot)

{

}

while (rightPointer > 0 && a[--rightPointer] > pivot)

{

}

if (leftPointer >= rightPointer)

{

break;

}

else swap(leftPointer, rightPointer);

}

swap(leftPointer, right);

return leftPointer;

}

void QuickSort(int left, int right)

{

if (right - left <= 0)

return;

else

{

int pivot = a[right];

int partition = partitionFunc(left, right, pivot);

QuickSort(left, partition - 1);

QuickSort(partition + 1, right);

}

}

## 7.5. Heap Sort

Dãy gọi là một HEAP nếu thỏa mãn và

Chú ý:

Các phần tử có chỉ số n/2,…n-1 là nút lá

HEAP có n phần tử thì có n/2 nút trong

void heapify(int arr[], int n, int i) { //Hàm để xây dựng max heap

int largest = i;

int left = 2 \* i + 1;

int right = 2 \* i + 2;

if (left < n && arr[left] > arr[largest])

largest = left;

if (right < n && arr[right] > arr[largest])

largest = right;

if (largest != i) {

swap(arr[i], arr[largest]);

heapify(arr, n, largest);

}

}

void HeapSort(int arr[], int n) { //Hàm heap

for (int i = n / 2 - 1; i >= 0; i--) {

heapify(arr, n, i);

}

for (int i = n - 1; i > 0; i--) {

swap(arr[0], arr[i]);

heapify(arr, i, 0);

}

}

Kết quả của các phép sắp xếp:

A screen shot of a black background

Description automatically generated

# CHƯƠNG VIII: TÌM KIẾM

Cho n bản ghi khóa tương ứng Ki, tìm bản ghi có giá trị khóa = x.

## 8.1. Tìm kiếm tuần tự (Sequential Search)

Ý tưởng: Lần lượt tìm kiếm từ bản ghi đầu tiên cho đến khi tìm thấy hoặc không còn phần tử để tìm kiếm. Thực hiện trên mảng hoặc danh sách liên kết đơn.

int Seq\_Search(int X, int a[], int n)

{

int i = 0;

while (i < n && a[i] != X)

{

i++;

}

if (i < n) return i;

return -1;

}

## 8.2. Tìm kiếm nhị phân (Binary Search)

Ý tưởng: Tiến hành treen dãy đã được sắp xếp tăng dần. Dãy tùm kiếm a[1], a[2],…,a[n], khóa tìm kiếm X.

So sánh X với a[(n+1) div 2]:

Nếu X = a[(n+1) div 2]: Thuật toán kết thúc

Nếu X < a[(n+1) div 2]: Quay lại tìm kiếm trong dãy a[1], a[2],…,a[(n+1) div 2-1]

Nếu X > a[(n+1) div 2]: Quay lại tìm kiếm trong dãy a[(n+1) div 2+1],…,a[n]

int Bin\_Search(int X, int a[], int n)

{

int l = 0, r = n - 1;

while (l <= r)

{

int m = (l + r) / 2;

if (X < a[m]) r = m - 1;

else if (X > a[m]) l = m + 1;

else return m;

}

return -1;

}

Kết quả của các phép tìm kiếm:

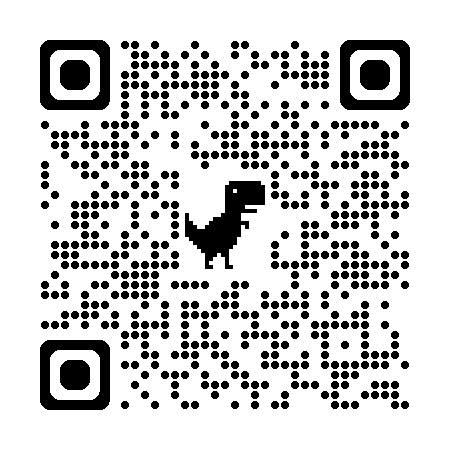
A screen shot of a computer

Description automatically generated

# CHƯƠNG IX: TỔNG KẾT

## 9.1. Mã nguồn chương trình

Do các chương trình rất dài nên em sẽ đưa lên nền tảng Github:



QR code dẫn tới chương trình

## 9.2. Tổng kết

Qua quá trình nghiên cứu và thực hiện tiểu luận môn Cấu trúc dữ liệu và Giải thuật, em đã hiểu sâu hơn về vai trò quan trọng của cấu trúc dữ liệu và thuật toán trong việc phát triển các giải pháp phần mềm hiệu quả. Các cấu trúc dữ liệu cơ bản như mảng, danh sách liên kết, ngăn xếp, hàng đợi, cây và đồ thị không chỉ giúp tổ chức và quản lý dữ liệu một cách tối ưu mà còn là nền tảng để xây dựng và áp dụng các thuật toán phức tạp hơn trong các lĩnh vực khác nhau. Qua đó, em đã nhận thức được tầm quan trọng của việc nắm vững các cấu trúc dữ liệu và thuật toán cơ bản như một bước đệm cần thiết cho sự phát triển các kỹ năng lập trình và giải quyết vấn đề trong lĩnh vực khoa học máy tính. Đây là những kỹ năng cốt lõi mà em sẽ tiếp tục phát triển và hoàn thiện trong quá trình học tập và làm việc sau này.