**TRƯỜNG ĐẠI HỌC KỸ THUẬT CÔNG NGHIỆP**

**KHOA ĐIỆN TỬ - BỘ MÔN CÔNG NGHỆ THÔNG TIN**

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

Description automatically generated with medium confidence**

**TIỂU LUẬN**

**HỌC PHẦN : CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT**

|  |  |  |
| --- | --- | --- |
| **GIÁO VIÊN HƯỚNG DẪN** | **:** | **Th.S Nguyễn Thị Hương** |
| **HỌ VÀ TÊN SINH VIÊN** | **:** | **Nguyễn Thị Thu Hiền** |
| **LỚP** | **:** | **K58KMT** |
|  |  |  |

**THÁI NGUYÊN - 2024**

**PHIẾU GIAO BÀI TẬP TIỂU LUẬN**

Môn cấu trúc dữ liệu & giải thuật

Lớp K58KMT, học kì 1 năm học 2024-2025

GV giảng dạy: Nguyễn Thị Hương ([huongktpm@tnut.edu.vn](mailto:huongktpm@tnut.edu.vn))

Thái nguyên, Ngày 29/11/2024

Chú ý: Kết quả chấm bài tập tiểu luận sẽ được sử dụng làm điểm đánh giá kết thúc học phần

I.Nội dung bài tập tiểu luận:

Đề tài môn học đã giao cho sinh viên thực hiện

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

II. Thời gian thực hiện:

III. Hình thức nộp:

1. Báo cáo bản word
2. File source code chương trình

**NHẬN XÉT CỦA GIÁO VIÊN**

Giáo viên chấm thi

*Ký tên*

**LỜI CAM ĐOAN**

Bài tiểu luận trên của em, môn học “*Cấu trúc dữ liệu và giải thuật*” là do chính cá nhân em làm. Em xin cam đoan, không sao chép, không trao đổi mua bán hoặc bất kỳ hình thức nào mà không tự bản thân thực hiện. Mong rằng thầy cô giáo sẽ xem xét kĩ lưỡng bài tiểu luận, để đưa ra đánh giá về tính xác thực!

Em xin trân thành cảm ơn!

Sinh viên thực hiện

# **LỜI NÓI ĐẦU**

Ngày nay, nền khoa học kỹ thuật ngày càng phát triển, đặc biệt là ngành công nghệ thông tin. ứng dụng của công nghệ đã tràn ngập mọi mặt trong đời sống. Từ vật chất như điện thoại, tivi, ... tới các mạng xã hội đều ứng dụng của công nghệ thông tin. Thế giới ngày càng hiện đại là do ngành công nghệ thông tin. Chính vì tầm quan trọng và đa dạng nên sinh viên cần phải nắm vững các kiến thức, nguyên lý , ứng dụng của các môn học cho ngành này. Môn học “Cấu trúc dữ liệu và giải thuật” là môn trang bị kiến thức cơ bản về giải thuật và các cách cài đặt chương trình, thiết kế dữ liệu. Trong phạm vi học phần, bao gồm các kiến thức về đệ quy , danh sách , mảng , các thuật toán tìm kiếm , sắp xếp , đồ thị... sẽ được nêu tổng quát , trong bài tiểu luận trên. Nếu có sai sót hoặc thiếu sót, em mong rằng quý thầy cô, các bạn góp ý để em có thể hoàn thiện tốt các bài làm của mình trong những lần sau.

Em xin trân thành cảm ơn!

**MỤC LỤC**

[**LỜI NÓI ĐẦU** 5](#_Toc182641490)

[**PHẦN MỞ ĐẦU** 8](#_Toc182641491)

[1. Giới thiệu về môn học 8](#_Toc182641492)

[2. Giới thiệu chung về bài tiểu luận 8](#_Toc182641493)

[**CHƯƠNG 1,2: TỔNG HỢP LÝ THUYẾT** 9](#_Toc182641494)

[2.1 Lý thuyết 9](#_Toc182641495)

[2.2 Trình bày nội dung giải thuật 9](#_Toc182641496)

[2.3 Cài đặt, kiểm nghiệm chương trình 10](#_Toc182641497)

[**CHƯƠNG 3: ĐỆ QUY** 10](#_Toc182641498)

[3.1 Lý thuyết 10](#_Toc182641499)

[3.2 Trình bày nội dung giải thuật 11](#_Toc182641500)

[3.3 Cài đặt, kiểm nghiệm chương trình 13](#_Toc182641501)

[**CHƯƠNG 4: MẢNG VÀ DANH SÁCH** 13](#_Toc182641502)

[4.1 Lý thuyết 13](#_Toc182641503)

[4.2 Trình bày nội dung giải thuật 14](#_Toc182641504)

[4.3 Cài đặt, kiểm nghiệm chương trình 15](#_Toc182641505)

[**CHƯƠNG 5: DANH SÁCH MÓC NỐI** 15](#_Toc182641506)

[5.1 Lý thuyết 15](#_Toc182641507)

[5.2 Trình bày nội dung giải thuật 18](#_Toc182641508)

[5.3 Cài đặt, kiểm nghiệm chương trình 23](#_Toc182641509)

[**CHƯƠNG 6: CÂY** 23](#_Toc182641510)

[6.1 Lý thuyết 23](#_Toc182641511)

[6.2 Trình bày nội dung giải thuật 25](#_Toc182641512)

[6.3 Cài đặt, kiểm nghiệm chương trình 27](#_Toc182641513)

[**CHƯƠNG 7: ĐỒ THỊ** 27](#_Toc182641514)

[7.1 Lý thuyết 27](#_Toc182641515)

[7.2 Trình bày nội dung giải thuật 31](#_Toc182641516)

[7.3 Cài đặt, kiểm nghiệm chương trình 36](#_Toc182641517)

[**CHƯƠNG 8: SẮP XẾP** 36](#_Toc182641518)

[8.1 Lý thuyết 36](#_Toc182641519)

[8.2 Trình bày nội dung giải thuật 40](#_Toc182641520)

[8.3 Cài đặt, kiểm nghiệm chương trình 44](#_Toc182641521)

[**CHƯƠNG 9: TÌM KIẾM** 44](#_Toc182641522)

[9.1 Lý thuyết 44](#_Toc182641523)

[9.2 Trình bày nội dung giải thuật 44](#_Toc182641524)

[9.3 Cài đặt, kiểm nghiệm chương trình 44](#_Toc182641525)

[**TỔNG KẾT** 44](#_Toc182641526)

[**PHỤ LỤC** 44](#_Toc182641527)

[**TÀI LIỆU THAM KHẢO** 45](#_Toc182641528)

# **PHẦN MỞ ĐẦU**

1. Giới thiệu về môn học

* Giải thuật ( statements)

Là một dãy các câu lệnh chặt chẽ và rõ ràng, xác định một trình tự các thao tác trên một số đối tượng nào đó, sao cho sau một số hữu hạn các bước thực hiện, cho ta một kết quả mong muốn

Giải thuật chỉ phản ánh các phép xử lý

* Dữ liệu ( Data)

Biểu diễn các thông tin cần thiết cho bài toán, là đối tượng để xử lý trên Máy tính

Các phần tử của dữ liệu thường có mối quan hệ với nhau, việc tổ chức dữ liệu theo một cấu trúc thích hợp (CTDL) giúp cho việc thực hiện các phép xử lý trên dữ liệu được thuận lợi đạt hiệu quả cao hơn.

1. Giới thiệu chung về bài tiểu luận

Nhằm mục đích củng cố lại toàn bộ kiến thức đã học của học phần “ Cấu trúc dữ liệu và giải thuật” , đưa ra các giải thuật cho những dạng bài liên quan theo từng chương, code chạy chương trình cho bài giải thuật đó.

Bài tiểu luận bao gồm các chương:

* Chương 1,2: Tổng hợp lý thuyết
* Chương 3: Đệ quy
* Chương 4: Mảng và danh sách
* Chương 5: Danh sách móc nối
* Chương 6: Cây
* Chương 7: Đồ thị
* Chương 8: Tìm kiếm
* Chương 9: Sắp xếp

# **CHƯƠNG 1,2: TỔNG HỢP LÝ THUYẾT**

## Lý thuyết

* Giới thiệu về phân tích, thiết kế giải thuật

Phân tích và thiết kế giải thuật là quá trình tìm ra một cách tiếp cận hiệu quả để giải quyết một bài toán cụ thể. Nó bao gồm các bước:

* Phân tích bài toán: Hiểu rõ yêu cầu của bài toán, xác định đầu vào, đầu ra và các ràng buộc.
* Thiết kế giải thuật: Xây dựng một chuỗi các bước cụ thể để giải quyết bài toán.
* Phân tích giải thuật: Đánh giá hiệu quả của giải thuật về thời gian và không gian.

Các khái niệm cơ bản

* Thuật toán: Một tập hợp hữu hạn các hướng dẫn rõ ràng, có thể thực hiện được, dùng để giải quyết một bài toán.
* Độ phức tạp thuật toán: Đo lường hiệu suất của một thuật toán về thời gian và không gian. Thường được biểu diễn bằng ký hiệu O lớn.
* Cấu trúc dữ liệu: Cách tổ chức dữ liệu để hỗ trợ các thao tác của thuật toán.
* Độ phức tạp không gian: Lượng bộ nhớ mà thuật toán sử dụng.
* Độ phức tạp thời gian: Thời gian thực thi của thuật toán.

Quá trình phân tích và thiết kế giải thuật

* Hiểu rõ bài toán:
* Xác định rõ đầu vào, đầu ra và các ràng buộc của bài toán.

Ví dụ: Bài toán tìm số lớn nhất trong một dãy số.

Đầu vào: Một dãy số.

Đầu ra: Số lớn nhất trong dãy.

Ràng buộc: Dãy số có thể rỗng hoặc chứa các số âm.

Xây dựng thuật toán:

Sử dụng các cấu trúc điều khiển (if-else, vòng lặp) và các cấu trúc dữ liệu phù hợp để biểu diễn thuật toán.

Ví dụ: Thuật toán tìm số lớn nhất.

Khởi tạo một biến max bằng phần tử đầu tiên của dãy.

Duyệt qua từng phần tử còn lại trong dãy.

Nếu phần tử hiện tại lớn hơn max thì cập nhật max.

Trả về max.

Phân tích thuật toán:

Đánh giá độ phức tạp thời gian và không gian của thuật toán.

Sử dụng các kỹ thuật phân tích để tìm ra các trường hợp tốt nhất, xấu nhất và trung bình của thuật toán.

Ví dụ: Thuật toán tìm số lớn nhất có độ phức tạp thời gian O(n), trong đó n là số phần tử của dãy.

## Trình bày nội dung giải thuật

Bài 1: Viết giải thuật cho bài toán đọc 2 số nguyên và in ra kết quả của phép (+), phép trừ (-), phép nhân (\*), phép chia (/) ra màn hình. Nhận xét kết quả chia 2 số nguyên.

\***Giải thuật**

|  |
| --- |
| int main(){  int a,b;  cout<<"nhap a,b: ";cin>>a>>b;  cout<<"\ncong="<<a+b;  cout<<"\ntru="<<a-b;  cout<<"\nnhan="<<a\*b;  cout<<"\nchia="<<float(a)/b; } |

## Cài đặt, kiểm nghiệm chương trình

*Bài1: Kết quả kiểm nghiệm*

A screenshot of a computer

Description automatically generated

Hình 2.3.1 Kết quả chương trình B1\_C1,2

# **CHƯƠNG 3: ĐỆ QUY**

## 3.1 Lý thuyết

Đệ quy là một kỹ thuật lập trình trong đó một hàm gọi lại chính nó để giải quyết một vấn đề. Nói cách khác, một hàm đệ quy là một hàm định nghĩa một phần hoặc toàn bộ về mặt chính nó.

* *Đặc điểm của thủ tục đệ quy* :

+ Trong thủ tục đệ quy có lời gọi đến chính nó.

+ Mỗi lần có lời gọi thì kích thước của bài toán đã thu nhỏ hơn trước

+ Có một trường hợp đặc biệt, trường hợp suy biến: Bài toán sẽ được giải quyết theo một cách khác hẳn và gọi đệ quy cũng kết thúc.

* *Đệ quy :*

+ Đệ quy trực tiếp (Directly recursive): Thủ tục chứa lời gọi đến chính nó

+ Đệ quy gián tiếp (indirectly recursive): Thủ tục chứa lời gọi đến thủ tục khác mà thủ tục này lại chứa lời gọi đến chính nó

* *Thiết kế giải thuật đệ quy:*

Hàm N! được định nghĩa:

A black text with a white background

Description automatically generated with medium confidence

* *Giải thuật đệ quy được viết dưới dạng thủ tục hàm như sau:*

int factorial(int n) {

if (n == 0) {

return 1;

} else {

return n \* factorial(n - 1); } }

## 3.2 Trình bày nội dung giải thuật

3.2.1 Dãy FIBONACI

Dãy này bắt nguồn từ bài toán cổ về việc sinh sản của các cặp thỏ theo những luật sau:

1) Các con thỏ không bao giờ chết

2)Hai tháng sau khi ra đời một cặp thỏ sẽ sinh ra một cặp thỏ con ( 1 đực, 1 cái)

3) Khi đã sinh con rồi, thì mỗi tháng tiếp chúng lại sinh ra được một cặp con mới

Giả sử bắt đầu từ một cặp mới ra đời thì đến tháng thứ n sẽ có bao nhiêu cặp ?

Với n = 6 => ta thấy:

Tháng 1: 1 cặp (Cặp ban đầu)

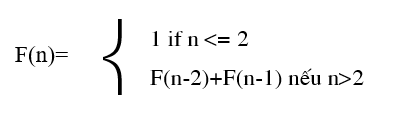
Tháng 2 : 1 cặp (Cặp ban đầu vẫn chưa đẻ)

Tháng 3: 2 cặp ( đã có thêm một cặp con)

Tháng 4: 5 cặp (Cặp con bắt đầu đẻ)

Trong các cặp thỏ ở tháng thứ n –1 chỉ có những cặp ở tháng thứ (n-2) mới sinh con ở tháng thứ n → F(n) = F(n-2) + F(n-1)

Vậy ta có thể viết F(n) theo công thức:



* *Giải thuật đệ quy cho dãy Fibonaci:*

int fibonacci(int n) {

if (n <= 2) {

return n;

} else {

return fibonacci(n - 1) + fibonacci(n - 2);

}

}

3.2.2 Bài toán “**Tháp Hà Nội**”

- Có n đĩa, kích thước nhỏ dần, đĩa có lỗ ở giữa. Có thể xếp chồng chúng lên nhau, xuyên qua một cặp, to dưới, nhỏ trên

- Yêu cầu:

Chuyển chồng đĩa từ cọc A sang cọc C, theo những điều kiện :

1) Mỗi lần chỉ được chuyển một đĩa

2) Không khi nào có tình huống, đĩa to ở trên đĩa nhỏ ở dưới (dù là tạm thời)

3) Được phép sử dụng một cọc trung gian (cọc B)

→ Xét các trường hợp:

\* Trường hợp một đĩa:

Chuyển từ cọc A sang cọc C

\* Trường hợp 2 đĩa:

Chuyển đĩa 1 từ cọc A sang cọc B

Chuyển đĩa 2 từ cọc A sang cọc C

Chuyển đĩa 1 từ cọc B sang cọc C

Như vậy :

Bài toàn tháp hà nội tổng quát với n đĩa được dẫn tới bài toán tương tự với kích thước nhỏ hơn. chẳng han: Chuyển n đĩa từ cọc A sang cọc C nay là chuyển n-1 đĩa từ cọc A sang cọc B ở mức này, giải thuật sẽ là:

- Chuyển (n-2) đĩa từ cọc A sang cọc C

- Chuyển 1 đĩa từ cọc A sang cọc B

- Chuyển (n-2) đĩa từ cọc C sang cọc B

Chuyển (n-1) đĩa từ cọc B sang cọc C => giải thuật sẽ là :

- Chuyển (n-2) đĩa từ cọc B sang cọc A

- Chuyển 1 đĩa từ cọc B sang cọc C

- Chuyển (n-2) đĩa từ cọc A sang cọc C

Cứ như thế cho tới khi trường hợp suy biến sảy ra, đó là trường hợp ứng với bài toán chuyển 1 đĩa mà thôi

* *Giải thuật đệ quy cho “Tháp Hà Nội”:*

Procedure HANOI( n, B, A, C)

*if* (n=1) Chuyen (A , C){}

*else*

HANOI (n-1, A, C, B) ;

HANOI(1, A, B, C);

HANOI(n-1, B, A, C);

*return*

3.2.3 Bài toán 8 Quân hậu

* Bài toán:

Dữ liệu vào :

Bàn cờ 8 ô

8 quân hậu

- Yêu cầu: Đặt 8 quân hậu vào 8 ô khác nhau trên bàn cờ, sao cho mỗi cột chỉ có 1 quân, mỗi đường chéo chỉ có một quân

- Ý tưởng của giải thuật:

Sử dụng phương pháp vét cạn (thử) mọi khả năng có thể có

Sử dụng 2 kỹ thuât đó là quay lui( lần ngược trở lại) và đệ quy

* Cách hoạt động:
* Bắt đầu đặt quân hậu từ hàng đầu tiên (row = 0).
* Thử đặt quân hậu vào từng ô trong hàng (cột từ 0 đến 7).
* Nếu ô đó an toàn (dựa trên hàng trước đó):
* Đặt quân hậu và gọi đệ quy để giải hàng tiếp theo.
* Nếu không còn vị trí hợp lệ ở hàng hiện tại, quay lại hàng trước đó và thử vị trí khác.

- Khi hoàn thành hàng cuối (row = 8), in ra một nghiệm.

## 3.3 Cài đặt, kiểm nghiệm chương trình

* Dãy Fibonaci

*Kết quả chương trình:*

A black screen with white text

Description automatically generatedHình 3.3.1 Kết quả chương trình dãy Fibonaci

* Bài toán “ Tháp Hà Nội”

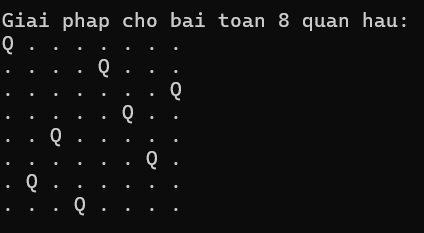
*Kết quả chương trình:*

# 

Hình 3.3.2 Kết quả chương trình bài toán “Tháp Hà Nội”

* Bài toán 8 quân hậu

*Kết quả chương trình:*



Hình 3.3.3 Kết quả chương trình bài toán “8 quân hậu”

# **CHƯƠNG 4: MẢNG VÀ DANH SÁCH**

## 4.1 Lý thuyết

4.1.1 Các khái niệm

*Mảng:* 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.

*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(d/s được sắp) :

Là một dãy gồm các phẩn tử x1, x2, ...., xn có cùng một kiểu dữ liệu trong đó: n được gọi là độ dài của d/s (số lượng phần tử của d/s), nếu n=0 thì gọi là danh sách rỗng

Nếu n>= 1 thì:

X1 gọi là phần tử đầu tiên

Xn gọi là phần tử cuối cùng

Các phép toán trên danh sách

Bổ sung vào danh sách một phần tử mới trước hoặc sau một phần tử nào đó thuộc d/s

- Loại bỏ khỏi danh sách một phần tử nào đó

- truy cập đến một phần tử nào đó thuộc d/s

- Tìm số lượng phần tử trong d/s

- Duyệt (đọc) d/s từ trái sang phải hoăc từ phải sang trái

Ngoài ra còn có các phép toán sau:

- Ghép 2 hoặc nhiều d/s

- Tách một d/s thành nhiều d/s

- Sao chép mọt d/s

- Cập nhật (update) d/s

- Sắp xếp các phần tử trong d/s theo một thứ tự nhất định

Tệp ( file):

Là một loại d/s có kích thước lớn, được lưu trữ ở bộ nhớ ngoài

Phần tử cơ bản của tệp là bản ghi(record) gồm n trường dữ liệu cần thiết; tương ứng với các thuộc tính khác nhau

* Cấu trúc lưu trữ 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)

A white rectangular object with black text

Description automatically generated

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 ai sẽ được tính bởi:

Loc(ai) =L0 + c\*(i-1)

F(i)=c\*(i-1) gọi là hàm địa chỉ ( addressfunction)

Loc(ai)= L0 +c\*(i-b)

Tương tự: Với mảng nhiều chiều, việc lưu trữ vẫn bằng véc tơ lưu trữ như trên

VD: Trong FORTRAN, ma trận 3 hàng 4 cột (aịj) với 1<=i<=3, 1<=j<=4 sẽ được lưu trữ kế tiếp như sau:



→ Lưu trữ ở đây là theo cột (gọi là thứ tự ưu tiên cột- column-major order)

Giả sử mỗi phần tử chiếm một từ máy thì địa chỉ của aijsẽ được tính bởi:

Loc(aij)=L0 +(j-1)\*3+(i-1)

Tổng quát:

Đối với ma trận n hàng, m cột→ Loc(aij)= L0 +(j-1)\*n+(i-1)

Chú ý:

Khi mảng được lưu trữ kế tiếp thì việc truy cập vào phần tử của mảng được thưc hiện trực tiếp dựa vào địa chỉ tính được, nên tốc độ nhanh và đồng đều đối với mọi phần tử

Mảng thể hiện mối quan hệ về cấu trúc giữa các phần tử dữ liệu ở nhiều ứng dụng. Song có nhiều trường hợp mảng cũng lộ ra những nhược điểm của nó

* Stack (ngăn xếp)

Ngăn xếp (Stack) là một d/s tuyến tính đặc biệt mà phép bổ sung và phép loại bỏ luôn luôn thực hiện ở một đầu, gọi là đỉnh(TOP)

Ví dụ:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1  111 | 2  222 | 4  333 | 3  444 |

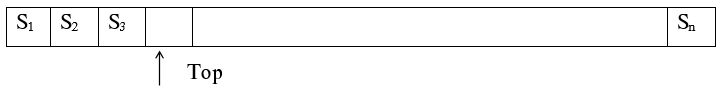
*Nguyên tắc:* “Vào sau ra trước”

(Last in first out: LIFO). Stack có thể rỗng hoặc bao gồm 1 số phần tử

* Lưu trữ stack bẳng mảng

Có thể lưu trữ Stack bằng một véc tơ lưu trữ S, gồm n phần tử nhớ kế tiếp

Cấu trúc lưu trữ của Stack có dạng :



T: con trỏ, trỏ tới đỉnh Stack, biến đổi khi stack hoạt động

T giảm 1: Khi loại bỏ 1 phần tử ra khỏi stack

T tăng 1 : Khi thêm một phần tử vào stack

T=0 : Khi stack rỗng

Sau đây là giải thuật bổ xung và loại bỏ đối với stack.

Procedure Push (S, T, X)

{Giải thuật này thực hiện bổ xung phần tử X→Stack lưu trữ bởi véc tơ S có nhiều phần tử. T là con trỏ→Stack}

1. {Xét xem stack có tràn không?}

if (T>=n) cout<< “Stack tran”;

{Chuyển con trỏ}

T=T+1;

1. {Bổ xung phần tử mới X}

S [T]=X;

**Function Pop (S, T)**

(Hàm này thực hiện việc loại bỏ phần tử ở đỉnh stack S đang trỏ bởi T. Phần tử bị loại bỏ sẽ được thu nhận và đưa ra ngoài.)

1. {Xét xem stack có cạn không? hiện tượng cạn xẩy ra khi stack =rỗng, không còn phần tử nào để loại nữa, in ra thông báo cạn và kết thúc}

if ( T<= 0 ) cout<< “Stack cạn”; ;

{Chuyển con trỏ}

T=T-1;

1. {Đưa phần tử bị loại ra}

Pop=S [T+1];

*\* Chú ý:*

1. Đối với thủ tục hàm F nào đó, F=a; return ⇔ return a;

→ta có thể vết gọn bước 2), 3) và 4) như sau: return (S [T+1] );

1. Xử lý với nhiều Stack

Khi phải xử lý nhiều stack cùng một lúc → Xảy ra tình trạng stack này bị tràn trong khi không gian lưu trữ cho stack khác vẫn còn trống (tràn cục bộ ). Làm thế nào để khắc phục?

Nếu là 2 stack : Không quy định kích thước tối đa cho từng stack nữa mà không gian nhớ sẽ được dùng chung. Ta đặt 2 stack ở 2 đầu sao cho hướng phát tiển của chúng ngược chiều nhau:

A white background with black and white clouds

Description automatically generated



Do đó hiện tượng tràn xảy ra chỉ khi toàn bộ không gian nhớ dành cho nó đã được dùng hết

Nhưng nếu số lượng stack từ 3 trở nên: Phải có một biện pháp linh hoạt hơn, chẳng hạn: Có 3 stack, lúc đầu không gian nhớ có thể chia làm 3, nếu có một stack náo đó phát triển nhanh bị tràn trước mà stack khác vẫn còn chỗ trống thì phải dồn chỗ cho nó bằng cách:

+ Đẩy stack đứng sau sang phải

+ Lùi Stack đó sang trái

Đáy của stack phải được phép di động và giải thuật loại, bổ sung phần tử đối với stack hoạt động theo kiểu này cũng phải thay đổi

A diagram of a stack

Description automatically generated

## 4.2 Trình bày nội dung giải thuật

* *Dạng bài ứng dụng Stack*

*Bài 1.* Cho số nguyên 39. Hãy biểu diễn giải thuật chuyển đổi số nguyên đó sang số nhị phân trên theo cơ chế hoạt động của Stack.

3910=1001112

Theo stack : vẽ các khối push, pop.

\***Giải thuật**

int D = 39;

stack<int> binaryStack;

while (D > 0) {

int r = D % 2;

binaryStack.push(r);

D /= 2; }

cout << "so nhi phan la: ";

while (!binaryStack.empty()) {

cout << binaryStack.top();

binaryStack.pop();

}

cout << endl;

return 0; }

\*Vẽ các khối push,pop

Push

1

1

1

0

0

1

Pop

1

0

0

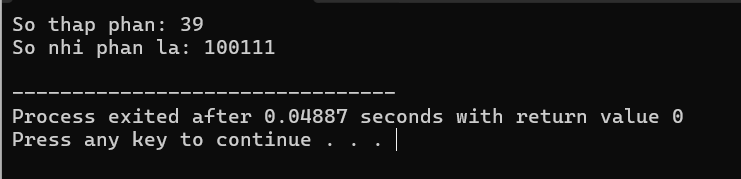
1

1

1

## 4.3 Cài đặt, kiểm nghiệm chương trình

*Bài 1: Kết quả chương trình*

**

Hình 4.3.1 Kết quả chương trình Stack

# **CHƯƠNG 5: DANH SÁCH MÓC NỐI**

## 5.1 Lý thuyết

Lưu trữ kế tiếp đối với danh sách tuyến tính bộc lộ nhiều nhược điểm khi thực hiện thường xuyên các phép bổ sung hoặc loại bỏ phần tử→Giải pháp nhằm khắc phục là sử dụng con trỏ hoặc mối nối để tổ chức danh sách tuyến tính

5.1.1 Danh sách móc nối đơn

* Nguyên tắc

Mỗi phần tử của d/s được lưu trữ trong một phần tử nhớ gọi là nút

* Mỗi nút bao gồm một số từ máy kế tiếp nhau
* Các nút có thế nằm ở vị trí bất kỳ trong bộ nhớ
* Mỗi nút chứa những thông tin ứng với một phần tử và địa chỉ phần tử đứng sau d/s:

A diagram of a link

Description automatically generated

Nút cuối cùng : Không có nút đứng sau → LINK chứa “Địa chỉ đặc biệt” dùng để đánh dấu nút kết thúc DS, ta gọi là mối nối không, ký hiệu NULL

Muốn truy nhập vào d/s ta phải truy nhập được vào phần tử đầu tiên, nghĩa là sử dụng con trỏ n trở tới node đầu tiên

*Hình ảnh d/s nối đơn:*

A diagram of a block diagram

Description automatically generated

*Quy ước:*D/s rỗng: L=NULL

Để tổ chức d/s móc nối, những khả năng sau đây phải có :

+ Tồn tại phương tiện để chia bộ nhớ ra thành các node, ở mỗi node có thể truy nhập váo từng trường (trường có kích thước ấn định)

+ Tồn tại một cơ chế xác định được một node đang sử dụng(nút bận), không sử dụng(Nút trống)

+ Tồn tại một cơ chế như một kho chứa chỗ trống, đêr cung cấp các node trống (list available space)

Để tiện trình bầy ta gọi cơ cấu trên là “DS chỗ trống” ta qui ước:

X<=VAIL: Cấp phát một nút có địa chỉ X, lấy từ d/s chỗ trống

X=> VAIL: Thu hồi một node có địa chỉ X trả về d/s chỗ trống

Ban đầu bản thân d/s chỗ trống là một stack nối đơn, AVAIL là con trỏ, trỏ tới nút đầu d/s, sau khi cấp phát và thu hồi ô nhớ => các node trong d/s không còn kế tiếp nhau nữa

* *Một số phép toán*

Xét một số giải thuật thể hiện một số phép toán thường tác động vào d/s nối đơn

-Với một node có đại chỉ p (được trỏ bởi P) thì :

+ INFO(P): Chỉ trường INFO của node ấy

+ LINK(P): Chỉ trường đại chỉ LINK của nó

1. Bổ sung một phần tử vào d/s nối đơn

Procedure INSERT(L, M, X)

{L: con trỏ, trỏ tới node đầu tiên của d/s

M:Con trỏ trỏ tới một node đang có trong d/s

Giải thuật: Bổ sung vào sau node trỏ bởi M một node có trường INFO lấy giá trị từ ô nhớ có đại chỉ X }

1. {Tạo node mới}

New<= AVAIL;

INFO(New)=X

1. {Thực hiện bổ sung, nếu d/s rỗng -> bổ sung vào d/s thành node đầu tiên, else bổ sung vào sau M}

if (L==null ) L=New;

LINK(null)=null

Else

LINK(New);

LINK(M)=New;

*Minh hoạ bởi hình ảnh sau:*

1. Nếu L=null thì sau phép bổ sung ta có:

A black arrow pointing to a black arrow

Description automatically generated

1. Nếu nút L<> null

A diagram of a block diagram

Description automatically generated

1. Loại bỏ một phần tử ra khỏi d/s nối đơn

Procedure DELETE(L, M)

{cho danh sách nối đơn trỏ bởi L. Giải thuật này thực hiện loại bỏ nút trỏ bởi M ra khỏi d/s đó}

1. {trường hợp d/s rỗng}

if (L=null) cout<<“d/s rỗng”;

1. {Trường hợp node trỏ bởi M là node đầu tiên trong d/s }

if (M=L) L==LINK(M);

M=> AVAIL;

1. {Tìm đến node đứng trước node trỏ bởi M }

P==L;

While (LINK(P)<M || LINK(P)> M ) do

P==LINK(P);

1. {Loại bỏ M}

LINK(P)==LINK(M);

1. {Đưa node bị loại về d/s chỗ rtống}

M==AVAIL;

1. Ghép 2 d/s nối đơn

Procedure COMBINE(P, Q) {Ghép 2 d/s nối đơn lần lượt trỏ bởi P, Q thành một d/s mới và cho P trỏ tới nó}

1. {Trường hợp Q rỗng}

if (Q= null) return Q

1. {p Rỗng }

if (p=null ) P==Q;

1. {Tìm đến node cuối d/s }

P1==P;

While (LINK(P1)<null || LINK(P1) > null)

do P1=LINK(P1)

1. {Ghép}

LINK(P1)==Q;

*Chú ý:*

Cách tổ chức móc nối như trên rõ ràng tỏ ra thích hợp, tuy nhiên cách cài đặt cũng có những nhược điểm nhất định

+ Chỉ có phần tử đầu tiên được truy cập trực tiếp, các phần tử khác chỉ được truy nhập sau khi truy nhập phần tử trước nó

+ Tốn bộ nhớ vì có thêm trường LINK ở mỗi node để lưu trữ điạ chỉ tiếp theo

5.1.2 Danh sách móc nối kép

Danh sách liên kết kép là danh sách mà mỗi phần tử trong danh sách có kết nối với 1 phần tử đứng trước và 1 phần tử đứng sau nó.

|  |
| --- |
| 2 |

|  |
| --- |
| 3 |

|  |
| --- |
| **4** |

|  |
| --- |
| 1 |



* Cấu trúc danh sách móc nối kép

struct Node {

int data; // Giá trị của nút

Node\* prev; // Con trỏ tới nút trước

Node\* next; // Con trỏ tới nút sau

// Constructor

Node(int value) {

data = value;

prev = nullptr;

next = nullptr;

}

};

## 5.2 Trình bày nội dung giải thuật

*Bài 1.* Cho một danh sách móc nối đơn có nút dầu danh sách trỏ bởi L, với các trường info trong các nút là các số nguyên dương. Trình bày giải thuật tách danh sách đó thành hai danh sách móc nối đơn mà trường info của mọi nút trong danh sách cùng là chẵn hoặc cùng lẻ.

* Ý tưởng

1. **Khởi tạo**: Tạo hai danh sách mới, một dành cho các số chẵn và một cho các số lẻ. Gọi hai danh sách này lần lượt là EvenList và OddList.

2. **Duyệt qua danh sách ban đầu**: Dùng một con trỏ để duyệt từng phần tử trong danh sách bắt đầu từ nút đầu tiên (được trỏ bởi L).

3. **Phân loại nút**:

* Với mỗi nút trong danh sách ban đầu, kiểm tra giá trị trường info.
* Nếu giá trị là **chẵn** (sử dụng phép toán info % 2 == 0), thêm nút đó vào danh sách EvenList.
* Nếu giá trị là **lẻ** (sử dụng phép toán info % 2 != 0), thêm nút đó vào danh sách OddList.

4.**Tiếp tục duyệt**: Chuyển đến nút tiếp theo và lặp lại quá trình cho đến khi duyệt hết danh sách ban đầu.

5. **Kết thúc**: Khi duyệt xong danh sách ban đầu, em sẽ có hai danh sách riêng biệt: một chứa các số chẵn và một chứa các số lẻ.

*\* Giải thuật*

// Giả định rằng cấu trúc Node và danh sách móc nối đơn đã được định nghĩa sẵn

struct Node {

int info;

Node\* next;

};

void splitList(Node\* L, Node\*& EvenList, Node\*& OddList) {

EvenList = nullptr;

OddList = nullptr;

while (L != nullptr) {

Node\* newNode = new Node;

newNode->info = L->info;

newNode->next = nullptr;

if (L->info % 2 == 0) {

// Thêm vào danh sách chẵn

if (EvenList == nullptr) {

EvenList = newNode;

} else {

Node\* temp = EvenList;

while (temp->next != nullptr) {

temp = temp->next;

}

temp->next = newNode;

}

} else {

// Thêm vào danh sách lẻ

if (OddList == nullptr) {

OddList = newNode;

} else {

Node\* temp = OddList;

while (temp->next != nullptr) {

temp = temp->next;

}

temp->next = newNode;

}

}

L = L->next; // Di chuyển đến nút tiếp theo

}

}

*Bài 2.* Cho một danh sách móc nối đơn có nút dầu danh sách trỏ bởi P, với các trường info trong các nút là các số nguyên dương.

a,Viết thủ tục thêm một nút mới vào cuối danh sách

* *Giải thuật*

void addToTail(Node\*& head, int value) {

Node\* newNode = new Node(value); // Tạo nút mới

if (head == nullptr) { // Nếu danh sách rỗng

head = newNode;

return;

}

Node\* temp = head; // Duyệt tới cuối danh sách

while (temp->next != nullptr) {

temp = temp->next;

}

temp->next = newNode; // Thêm nút mới vào cuối

}

b,Viết thủ tục xoá một nút có trường info = x (x là giá trị cho trước) trong danh sách

* *Giải thuật*

void deleteNode(Node\*& head, int x) {

if (head == nullptr) return; // Nếu danh sách rỗng

// Trường hợp nút đầu chứa giá trị x

if (head->info == x) {

Node\* temp = head;

head = head->next;

delete temp;

return;

}

Node\* current = head;

// Duyệt tìm nút có giá trị x

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

current = current->next;

}

if (current->next == nullptr) return; // Không tìm thấy x

Node\* temp = current->next; // Nút cần xóa

current->next = temp->next; // Cập nhật con trỏ

delete temp*; // Giải phóng bộ nhớ*

*}*

## 5.3 Cài đặt, kiểm nghiệm chương trình

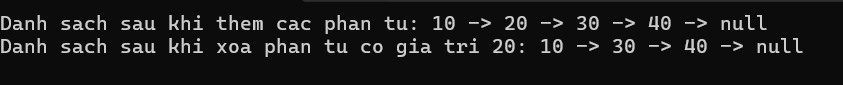
*Bài 1: Kết quả chương trình*

A number on a black background

Description automatically generated

Hình 5.3.1 Kết quả bài 1\_Danh sách móc nối

*Bài 2: Kết quả chương trình*



Hình 5.3.2 Kết quả bài 2\_Danh sách móc nối

# **CHƯƠNG 6: CÂY**

## 6.1 Lý thuyết

6.1.1 Các khái niệm

* Khái niệm về cây nhị phân:

Cây nhị phân (Binary Tree) là một cấu trúc dữ liệu dạng cây, trong đó mỗi nút có tối đa **hai** nút con. Các nút con được phân biệt thành “nút con trái” và “nút con phải”.

* *Nút gốc* (root): Là nút bắt đầu của cây.
* *Nút lá* (leaf): Là các nút không có nút con nào.
* *Nút cha* (parent) và *nút con* (child): Nếu nút A có nút con là B, thì A được gọi là nút cha của B và B là nút con của A.
* Các tính chất của cây nhị phân:
* *Số nút con*: Mỗi nút trong cây nhị phân có tối đa 2 nút con, gồm nút con trái và nút con phải.
* *Độ cao của cây*: Độ cao của cây nhị phân là số cạnh từ nút gốc đến nút lá xa nhất.
* *Số nút tối đa ở mức thứ k*: Ở mức thứ k, cây nhị phân có tối đa 2k2^k2k nút.

*Ví dụ:* Ở mức 0 có tối đa 20=12^0 = 120=1 nút (nút gốc), ở mức 1 có tối đa 21=22^1 = 221=2 nút, và cứ tiếp tục như vậy.

* *Số nút tối đa của cây nhị phân có độ cao h*: Cây nhị phân đầy đủ có số nút tối đa là 2h+1−12^{h+1} - 12h+1−1, với h là độ cao của cây.
* *Cây nhị phân đầy đủ* (Full Binary Tree): Là cây nhị phân mà mỗi nút hoặc là nút lá, hoặc có đủ hai nút con.
* *Cây nhị phân hoàn chỉnh* (Complete Binary Tree): Là cây nhị phân mà các nút trên mỗi mức được lấp đầy từ trái sang phải, chỉ có thể thiếu nút ở mức cuối cùng.
* *Cây nhị phân cân bằng* (Balanced Binary Tree): Là cây nhị phân mà độ cao của các cây con trái và cây con phải của mọi nút chỉ chênh lệch không quá 1.
* Các loại cây nhị phân đặc biệt**:**
* *Cây nhị phân tìm kiếm (Binary Search Tree - BST):* Là cây nhị phân mà mọi nút con bên trái có giá trị nhỏ hơn nút cha, và mọi nút con bên phải có giá trị lớn hơn nút cha.
* *Cây nhị phân hoàn hảo (Perfect Binary Tree):* Là cây nhị phân mà tất cả các mức đều đầy đủ, tức là mọi nút đều có đủ hai nút con, và mọi nút lá đều nằm ở cùng một mức.
* Một số ứng dụng của cây nhị phân:
* *Cây nhị phân tìm kiếm:* Được sử dụng trong các bài toán tìm kiếm và sắp xếp.
* *Cấu trúc biểu thức toán học*: Cây nhị phân có thể được sử dụng để biểu diễn các biểu thức toán học (cây biểu thức).
* *Lưu trữ tệp:* Trong hệ thống tệp, các thư mục và tệp có thể được tổ chức theo dạng cây nhị phân.
* Có các phép duyệt cây :
* Duyệt cây con thứ tự trước
* Duyệt cây con thứ tự giữa
* Duyệt cây con thứ tự sau

## 6.2 Trình bày nội dung giải thuật

* Cấu trúc duyệt cây nhị phân

struct Node {

int data; // Giá trị của nút

Node\* left; // Con trỏ đến cây con trái

Node\* right; // Con trỏ đến cây con phải

Node(int value) {

data = value;

left = right = nullptr;

}

};

* Duyệt cây nhị phân theo thứ tự trước

void preOrder(Node\* root) {

if (root == nullptr) return; // Điều kiện dừng

cout << root->data << " "; // Thăm nút gốc

preOrder(root->left); // Duyệt cây con trái

preOrder(root->right); // Duyệt cây con phải

}

* Duyệt cây theo thứ tự giữa

void inOrder(Node\* root) {

if (root == nullptr) return; // Điều kiện dừng

inOrder(root->left); // Duyệt cây con trái

cout << root->data << " "; // Thăm nút gốc

inOrder(root->right); // Duyệt cây con phải

}

* Duyệt cây theo thứ tự sau

void postOrder(Node\* root) {

if (root == nullptr) return; // Điều kiện dừng

postOrder(root->left); // Duyệt cây con trái

postOrder(root->right); // Duyệt cây con phải

cout << root->data << " "; // Thăm nút gốc

}

6.3 Cài đặt, kiểm nghiệm chương trình

*Bài 1.*Vẽ cây nhị phân biểu diễn biểu thức sau đây và viết lại biểu thức dưới dạng hậu tố: a\*(b+c)/(d\*e)

* Biểu thức dưới dạng hậu tố:

Theo kiểu duyệt trái, phải, gốc

A b c + \* d e \* /

A diagram of a network

Description automatically generated

*Kết quả chương trình*



Hình 6.2.1 Kết quả chương trình bài 1\_Cây

*Bài 2*:Vẽ cây nhị phân biểu diễn biểu thức sau đây. Hãy viết thứ tự các nút được thăm khi duyệt cây theo thứ tự trước? Theo thứ tự giữa? Theo thứ tự sau?

a\*(b+c)/(d\*e)

Trước: / \* a + b c \* d, e

Giữa: a \* b + c / d \* e

Sau: a b c + \* d e \* /

* *Kết quả chương trình*

A black background with white text

Description automatically generated

Hình 6.2.2 Kết quả chương trình bài 2\_Cây

# **CHƯƠNG 7: ĐỒ THỊ**

## 7.1 Lý thuyết

7.1.1 Các khái niệm

* Khái niệm đồ thị trong cấu trúc dữ liệu:

Đồ thị (Graph) là một cấu trúc dữ liệu dùng để biểu diễn các mối quan hệ giữa các đối tượng. Đồ thị được định nghĩa bởi một tập hợp các đỉnh (vertices) và một tập hợp các cạnh (edges) nối các cặp đỉnh với nhau.

* Các thành phần cơ bản của đồ thị:

1. Đỉnh (Vertex): Là một đối tượng trong đồ thị. Các đỉnh được dùng để đại diện cho các điểm hoặc các đối tượng mà ta muốn mô tả.
   * Ví dụ: Trong mạng xã hội, mỗi người dùng là một đỉnh.
2. Cạnh (Edge): Là một đường nối giữa hai đỉnh trong đồ thị, biểu diễn mối quan hệ giữa hai đối tượng. Cạnh có thể có hướng hoặc không có hướng.
   * Ví dụ: Trong mạng xã hội, mối quan hệ bạn bè giữa hai người có thể được biểu diễn bằng một cạnh.
3. Đồ thị có hướng (Directed Graph): Là đồ thị mà các cạnh có hướng, tức là từ một đỉnh này đến một đỉnh khác.
   * Ví dụ: Mối quan hệ theo dõi trên mạng xã hội (A theo dõi B nhưng B không cần theo dõi A).
4. Đồ thị vô hướng (Undirected Graph): Là đồ thị mà các cạnh không có hướng, tức là mối quan hệ giữa hai đỉnh là hai chiều.
   * Ví dụ: Quan hệ bạn bè trên mạng xã hội, nếu A là bạn của B thì B cũng là bạn của A.
5. Trọng số của cạnh (Weight): Trong một số đồ thị, mỗi cạnh có thể có một trọng số gắn kèm, biểu diễn độ dài, chi phí hoặc một giá trị nào đó.
   * Ví dụ: Đồ thị biểu diễn bản đồ, trọng số trên các cạnh có thể là khoảng cách giữa các địa điểm.

Có hai cách phổ biến để biểu diễn đồ thị:

1. Danh sách kề (Adjacency List):
   * Mỗi đỉnh sẽ có một danh sách chứa tất cả các đỉnh kề với nó. Cách này tiết kiệm không gian khi đồ thị thưa (có ít cạnh).
   * Ví dụ: Đỉnh A có danh sách chứa các đỉnh kết nối với A.
2. Ma trận kề (Adjacency Matrix):
   * Là một ma trận vuông với kích thước n×nn \times nn×n (n là số đỉnh). Mỗi phần tử A[i][j]A[i][j]A[i][j] trong ma trận sẽ là 1 nếu có cạnh giữa đỉnh iii và jjj, ngược lại là 0. Nếu đồ thị có trọng số, phần tử này sẽ lưu trọng số của cạnh.
   * Cách này phù hợp cho đồ thị dày đặc (nhiều cạnh).

* Các thuật toán quan trọng liên quan đến đồ thị:

1. Thuật toán tìm kiếm theo chiều sâu (Depth First Search - DFS):
   * Duyệt đồ thị bằng cách đi sâu vào các nhánh con trước khi quay lại.
   * Dùng để phát hiện chu trình, tìm tất cả các thành phần liên thông, và giải các bài toán đường đi.
2. Thuật toán tìm kiếm theo chiều rộng (Breadth First Search - BFS):
   * Duyệt đồ thị theo từng lớp, duyệt tất cả các đỉnh kề trước khi đi đến các đỉnh sâu hơn.
   * Thường được sử dụng để tìm đường đi ngắn nhất trong đồ thị vô hướng.
3. Thuật toán Dijkstra:
   * Dùng để tìm đường đi ngắn nhất từ một đỉnh nguồn đến các đỉnh khác trong đồ thị có trọng số dương.
4. Thuật toán Floyd-Warshall:
   * Dùng để tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh trong đồ thị.
5. Thuật toán Kruskal và Prim:
   * Dùng để tìm cây khung nhỏ nhất (Minimum Spanning Tree - MST) trong một đồ thị vô hướng có trọng số.

## 7.2 Trình bày nội dung giải thuật

*Các cách duyệt đồ thị*

* Duyệt theo chiều sâu
* Xuất phát từ đỉnh V . Xét 1 đỉnh w là lân cận của V nhưng chưa được thăm.
* Nếu 1 đỉnh V đã được thăm mà mọi đỉnh lân cận đều được thăm thì ta sẽ thăm lại cho đến khi không còn đỉnh nào không được thăm nữa.

**Giải thuật :**

Produce DEF(v)

VISITED(v)=1 ;

For(mỗi đỉnh w lân cận của v)

Do

If(visited(w)==0)

Def(w) ;

Return ;

* Duyệt theo chiều rộng
* Chọn 1 đỉnh xuất phát, giả sử là v ta lần lượt thăm tất cả các lân cận của v(duyệt mở rộng ra)

**Giải thuật :**

Produce BFS(v)

Visited(v)=1

Khoi tao q voi v da duoc nap vao

While( q rong)

{

Call CQDELETE(v,q)

}

For(w lan can v)

If (visited(w)==0)

{ call CQINSERT (w,q) ;

Visited(w)=1 ; }

} return ;

*Ví dụ:* Cây khung

* T bao gồm các cung thuộc 1 phép duyệt cây từ 1 đỉnh còn lại trong v
* Giá của cây khung T= tổng trọng số của các cung thuộc E
* Khung nào tạo thành 1 chu trình T thì loại bỏ cung đó(Chu trình là 1 đường đi đơn và khép kín)
* Trong 1 đồ thị có thể có nhiều cây khung, có thể theo chiều rộng, chiều sâu
* Dồ thị n đỉnh thì có n-1 cạnh
* Các cây khung không tạo nên chu trình

A network diagram with circles and lines

Description automatically generated

* Đồ thị G

A black and white drawing of a network

Description automatically generated

* Cây khung T của đồ thị G
* Cây khung cực tiểu

+ Là cây khung có trọng số cực tiểu

## Cài đặt, kiểm nghiệm chương trình

*Bài 1:* Cho đồ thị có hướng như hình vẽ. Hãy lập ma trận lân cận của đồ thị

*Đồ thị có hướng:*

* Gồm 8 đỉnh được đánh số từ 1 đến 8.
* Các cung có hướng nối các đỉnh, thể hiện mối quan hệ giữa chúng.

*Ma trận kề:*

* Là ma trận vuông kích thước n×nn \times nn×n, trong đó:
  + nnn: số đỉnh của đồ thị (ở đây n=8n = 8n=8).
  + A[i][j]=1A[i][j] = 1A[i][j]=1 nếu có cung đi từ đỉnh iii đến đỉnh jjj.
  + A[i][j]=0A[i][j] = 0A[i][j]=0 nếu không có cung nào nối từ iii đến jjj.

1 2 3 4 5 6 7 8

1 [ 0, 1, 1, 1, 0, 0, 0, 0 ]

2 [ 0, 0, 0, 0, 1, 0, 0, 0 ]

3 [ 0, 0, 0, 0, 1, 0, 0, 0 ]

4 [ 0, 0, 0, 0, 1, 0, 0, 0 ]

5 [ 0, 0, 0, 0, 0, 1, 1, 1 ]

6 [ 0, 0, 0, 0, 0, 0, 0, 1 ]

7 [ 0, 0, 0, 0, 0, 0, 0, 1 ]

8 [ 0, 0, 0, 0, 0, 0, 0, 0 ]

*Kết quả chương trình*

A screen shot of a number

Description automatically generated

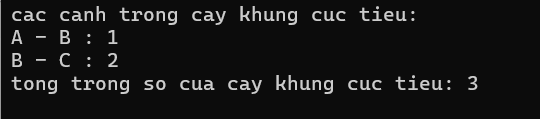
Hình 7.3.1 Kết quả chương trình bài 1\_Đồ thị

*Bài 2:* Cho đồ thị vô hướng có các trọng số lần lượt: (A,B)=1 ; (B,C)=2 ; (A,C)=3. Tính tổng trọng số của cây khung có giá trị cực tiểu.

\*Ý tưởng  
- Sử dụng thuật toán PRIM , chọn 1 đỉnh rồi duyệt các cạnh tới đỉnh kề nó .  
+ Đỉnh A : AB=1,AC=3 . Chọn AB =1 vì có trọng số nhỏ nhất   
+ Duyệt tiếp đỉnh A,B : BC=2,AC=3. Chọn BC có trọng số nhỏ nhất  
+ Duyệt tiếp đỉnh A,B,C có CA=3 tạo chu trình nên loại bỏ khỏi khung.

=> Vậy trọng số của cây khung nhỏ nhất : 1+2=3

*Kết quả chương trình*



Hình 7.3.2 Kết quả chương trình bài 2\_Đồ thị

# **CHƯƠNG 8: SẮP XẾP**

## 8.1 Lý thuyết

- *Sắp xếp* là quá trình bố trí lại các phần tử của một tập đối tượng nào đó, theo một thứ tự ấn định

- *Mục đích*: Giúp cho việc tìm kiếm được thuận lợi hơn

- n đối tượng được sắp ở đây ta quy ước là n bản ghi (record) mỗi bản ghi gồm một số trường (fields), dữ liệu tương ứng với những thuộc tính (attributs) khác nhau

ví dụ:

\* Bài toán sắp xếp:

Input: n đối tượng tương đương n khoá sắp xếp

Output: n đối tượng nhưng được sắp xếp theo trật tự mới ( tăng hoặc giảm)

\* Có 2 loại sắp xếp chính:

Sắp xếp trong (internal sorting): Là loại sắp xếp mà các đối tượng cần sắp xếp đã có đầy đủ, đồng thời trong bộ nhớ RAM

(khoá: Đại diện cho các bản ghi, để đơn giản ta chỉ nói tới giá trị khoá thôi )

=> Bài toán quy về: Dữ liệu vào: n số

Dữ liệu ra: n số theo trật tự mới

* Các kiểu sắp xếp và các phương pháp

- Sắp xếp chọn

- Sắp xếp chèn

- Sắp xếp đổi chỗ

+ Phương pháp nổi bọt

+ Phương pháp chèn

+ Phương pháp lựa chọn

+ Phương pháp hòa nhập

+ Phương pháp vun đống

+ Phương pháp sắp xếp nhanh

## 8.2 Trình bày nội dung giải thuật

## 8.3 Cài đặt, kiểm nghiệm chương trình

Hãy minh họa từng bước việc sắp xếp tăng dần dãy số sau theo giải thuật sắp xếp nổi bọt?

1 3 9 0 2 4 6 8 7 5

**Bước 1: Duyệt 1**

* So sánh 1 và 3: Không đổi.
* So sánh 3 và 9: Không đổi.
* So sánh 9 và 0: Đổi 9 và 0 → 1, 3, 0, 9, 2, 4, 6, 8, 7, 5
* So sánh 9 và 2: Đổi 9 và 2 → 1, 3, 0, 2, 9, 4, 6, 8, 7, 5
* So sánh 9 và 4: Đổi 9 và 4 → 1, 3, 0, 2, 4, 9, 6, 8, 7, 5
* So sánh 9 và 6: Đổi 9 và 6 → 1, 3, 0, 2, 4, 6, 9, 8, 7, 5
* So sánh 9 và 8: Đổi 9 và 8 → 1, 3, 0, 2, 4, 6, 8, 9, 7, 5
* So sánh 9 và 7: Đổi 9 và 7 → 1, 3, 0, 2, 4, 6, 8, 7, 9, 5
* So sánh 9 và 5: Đổi 9 và 5 → 1, 3, 0, 2, 4, 6, 8, 7, 5, 9
* Kết thúc duyệt 1: 1, 3, 0, 2, 4, 6, 8, 7, 5, 9

**Bước 2: Duyệt 2**

* So sánh 1 và 3: Không đổi.
* So sánh 3 và 0: Đổi 3 và 0 → 1, 0, 3, 2, 4, 6, 8, 7, 5, 9
* So sánh 3 và 2: Đổi 3 và 2 → 1, 0, 2, 3, 4, 6, 8, 7, 5, 9
* So sánh 3 và 4: Không đổi.
* So sánh 4 và 6: Không đổi.
* So sánh 6 và 8: Không đổi.
* So sánh 8 và 7: Đổi 8 và 7 → 1, 0, 2, 3, 4, 6, 7, 8, 5, 9
* So sánh 8 và 5: Đổi 8 và 5 → 1, 0, 2, 3, 4, 6, 7, 5, 8, 9
* Kết thúc duyệt 2: 1, 0, 2, 3, 4, 6, 7, 5, 8, 9

... *(Tiếp tục các bước duyệt cho đến khi không còn hoán đổi)*

Bài 2. Hãy minh họa từng bước việc sắp xếp tăng dần dãy số sau theo giải thuật sắp xếp chèn?

0 5 2 6 9 3 4 8 7 1

Bước 1: Chọn phần tử 5 (phần tử thứ 2)

* So sánh 5 với 0. Vì 5 lớn hơn 0, không cần di chuyển.
* Kết quả: 0, 5, 2, 6, 9, 3, 4, 8, 7, 1

Bước 2: Chọn phần tử 2 (phần tử thứ 3)

* So sánh 2 với 5: Vì 2 nhỏ hơn 5, di chuyển 5 sang phải.
* So sánh 2 với 0: Vì 2 lớn hơn 0, chèn 2 vào vị trí của 5.
* Kết quả: 0, 2, 5, 6, 9, 3, 4, 8, 7, 1

Bước 3: Chọn phần tử 6 (phần tử thứ 4)

* So sánh 6 với 5: Vì 6 lớn hơn 5, không cần di chuyển.
* Kết quả: 0, 2, 5, 6, 9, 3, 4, 8, 7, 1

Bước 4: Chọn phần tử 9 (phần tử thứ 5)

* So sánh 9 với 6: Vì 9 lớn hơn 6, không cần di chuyển.
* Kết quả: 0, 2, 5, 6, 9, 3, 4, 8, 7, 1

Bước 5: Chọn phần tử 3 (phần tử thứ 6)

* So sánh 3 với 9: Di chuyển 9 sang phải.
* So sánh 3 với 6: Di chuyển 6 sang phải.
* So sánh 3 với 5: Di chuyển 5 sang phải.
* So sánh 3 với 2: Vì 3 lớn hơn 2, chèn 3 vào vị trí của 5.
* Kết quả: 0, 2, 3, 5, 6, 9, 4, 8, 7, 1

Bước 6: Chọn phần tử 4 (phần tử thứ 7)

* So sánh 4 với 9: Di chuyển 9 sang phải.
* So sánh 4 với 6: Di chuyển 6 sang phải.
* So sánh 4 với 5: Di chuyển 5 sang phải.
* So sánh 4 với 3: Vì 4 lớn hơn 3, chèn 4 vào vị trí của 5.
* Kết quả: 0, 2, 3, 4, 5, 6, 9, 8, 7, 1

Bước 7: Chọn phần tử 8 (phần tử thứ 8)

* So sánh 8 với 9: Di chuyển 9 sang phải.
* So sánh 8 với 6: Vì 8 lớn hơn 6, chèn 8 vào vị trí của 9.
* Kết quả: 0, 2, 3, 4, 5, 6, 8, 9, 7, 1

Bước 8: Chọn phần tử 7 (phần tử thứ 9)

* So sánh 7 với 9: Di chuyển 9 sang phải.
* So sánh 7 với 8: Di chuyển 8 sang phải.
* So sánh 7 với 6: Vì 7 lớn hơn 6, chèn 7 vào vị trí của 8.
* Kết quả: 0, 2, 3, 4, 5, 6, 7, 8, 9, 1

Bước 9: Chọn phần tử 1 (phần tử thứ 10)

* So sánh 1 với 9: Di chuyển 9 sang phải.
* So sánh 1 với 8: Di chuyển 8 sang phải.
* So sánh 1 với 7: Di chuyển 7 sang phải.
* So sánh 1 với 6: Di chuyển 6 sang phải.
* So sánh 1 với 5: Di chuyển 5 sang phải.
* So sánh 1 với 4: Di chuyển 4 sang phải.
* So sánh 1 với 3: Di chuyển 3 sang phải.
* So sánh 1 với 2: Di chuyển 2 sang phải.
* So sánh 1 với 0: Vì 1 lớn hơn 0, chèn 1 vào vị trí của 2.
* Kết quả cuối cùng: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

# **CHƯƠNG 9: TÌM KIẾM**

* 1. Lý thuyết

Bài 1 :Nêu ý tưởng cơ bản của phương pháp tìm kiếm tuần tự trên một danh sách đã sắp xếp

Ý tưởng cơ bản:

1. Bắt đầu từ phần tử đầu tiên: Khởi đầu tại chỉ số đầu tiên của danh sách.
2. So sánh: So sánh phần tử hiện tại với giá trị cần tìm.
3. Tìm thấy: Nếu phần tử hiện tại bằng với giá trị cần tìm, trả về chỉ số của phần tử đó.
4. Chưa tìm thấy: Nếu không, di chuyển đến phần tử tiếp theo và lặp lại bước 2.
5. Kết thúc: Nếu đã duyệt hết danh sách mà không tìm thấy phần tử, trả về giá trị cho biết không tìm thấy (thường là -1 hoặc null).

## Trình bày nội dung giải thuật

* 1. Cài đặt, kiểm nghiệm chương trình

# **TỔNG KẾT**

# **PHỤ LỤC**

# **TÀI LIỆU THAM KHẢO**