**ĐẠI HỌC HUẾ**

# KHOA KỸ THUẬT VÀ CÔNG NGHỆ

🙠🙟🕮🙝🙢

****

**BÁO CÁO** **ĐỒ ÁN**

**NĂM HỌC 2020-2021**

**Giảng viên hướng dẫn: HỒ QUỐC DŨNG**

**Lớp: KHDL & TTNT**

|  |
| --- |
| Số phách  *(Do hội đồng chấm thi ghi)* |

**Thừa Thiên Huế, ngày …tháng…năm.....**

**ĐẠI HỌC HUẾ**

# KHOA KỸ THUẬT VÀ CÔNG NGHỆ

🙠🙟🕮🙝🙢

****

(MẪU BÌA PHỤ)

**BÁO CÁO** **ĐỒ ÁN**

**NĂM HỌC 2020-2021**

**Giảng viên hướng dẫn: HỒ QUỐC DŨNG**

**Lớp: KHDL & TTNT**

**Sinh viên thực hiện: HOÀNG TRỌNG TOÀN**

*(ký tên và ghi rõ họ tên)*

|  |
| --- |
| Số phách  *(Do hội đồng chấm thi ghi)* |

**Thừa Thiên Huế, ngày …tháng…năm.....**

# 

[KHOA KỸ THUẬT VÀ CÔNG NGHỆ](#_lf28s6t2mi65) 1

[KHOA KỸ THUẬT VÀ CÔNG NGHỆ](#_c287uhqssw3k) 2

[THÁP HÀ NỘI](#_bkrxbzmohnog) 3

[ƯỚC SỐ CHUNG NHỎ NHẤT](#_gk7wafu5pcvl) 4

[TÍNH GIAI THỪA](#_o951xtv1bx6) 5

[MÃ ĐI TUẦN](#_ll136k8u3enm) 12

[8 QUÂN HẬU](#_l7ey1i6gfy1e) 14

[Bài toán 8 con Hậu là bài toán nổi tiếng trong lĩnh vực toán học. Trong bài toán này thiết lập thường xuyên được nhắc nhở khi có liên quan đến chủ đề và đệ trình quay lui hoặc là trí tuệ nhân tạo.](#_v1vtrfeyf4hd) 14

[DANH SÁCH LIÊN KẾT ĐƠN](#_crv6ic8yrl29) 14

[DANH SÁCH LIÊN KẾT KÉP](#_se35mx26i0d6) 21

[NGĂN XẾP STACK](#_3n1st25fs35t) 22

[HÀNG ĐỢI QUEUE](#_kt2le2ylcybu) 24

[DUYỆT CÂY TRƯỚC ĐẾN SAU VÀ SAU ĐẾN TRƯỚC](#_yd38gj7qginj) 25

[ĐỒ THỊ VÔ HƯỚNG](#_iqp2gkdgyztk) 25

[ĐỒ THỊ CÓ HƯỚNG](#_qhy6yxpewroh) 26

[SELECTION SORT](#_1ak4gh5mtyw0) 26

[INSERTION SORT](#_5kae8p733918) 33

[BUBBLE SORT](#_wsianbo0vl8b) 34

[QUICK SORT](#_8fyktpx5n22g) 35

[HEAP SORT](#_f8az3j8n45bq) 36

[MERGE SORT](#_1adrd6levyzv) 37

# 

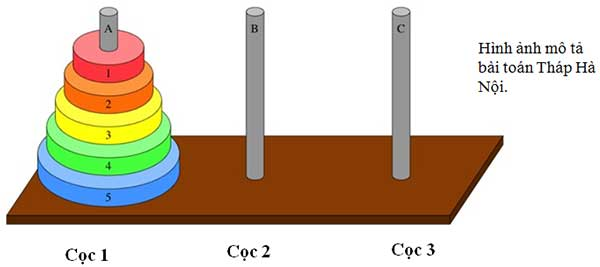
# 

# 

# THÁP HÀ NỘI

Bài toán tháp Hà Nội là trò chơi toán học gồm 3 cọc và n đĩa có kích thước khác nhau. Ban đầu các đĩa được xếp chồng lên nhau trong cọc A như hình vẽ.

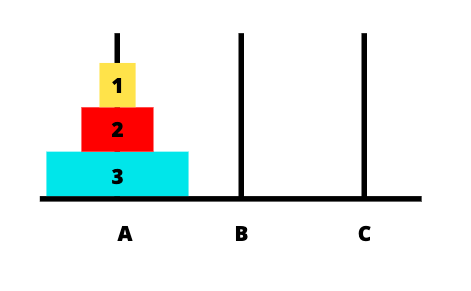
Yêu cầu của bài toán: Di chuyển toàn bộ các đĩa ở cọc A sang cọc C với điều kiện sau.



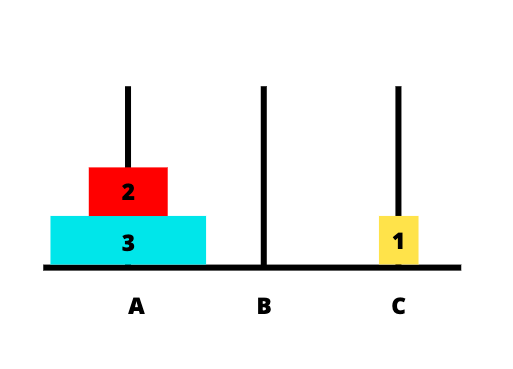
* Mỗi lần thực hiện chỉ được di chuyển một đĩa
* Các đĩa phải xếp theo nguyên tắc, đĩa lớn ở dưới, đĩa nhỏ ở trên.
* Được phép thêm một cọc B làm trung gian để di chuyển các đĩa.

Để dễ hình dung hơn các bước giải, chúng ta sẽ lấy ví dụ cụ thể. Giả sử chúng ta có 3 cọc A, B, C tương trưng cho 3 tháp và có 3 đĩa.

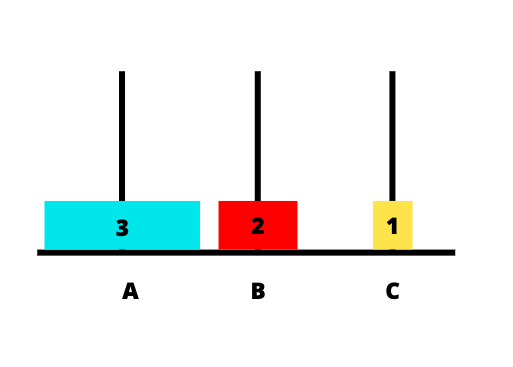
Bắt đầu



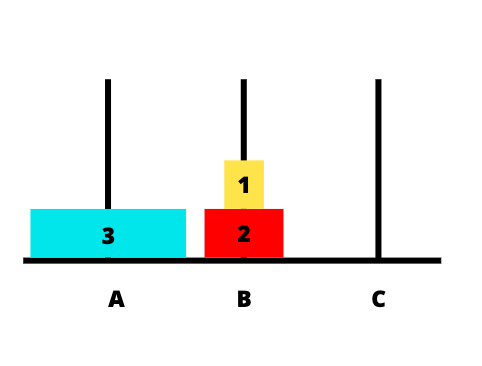
Di chuyển đĩa 1 từ cọc A sang cọc C.



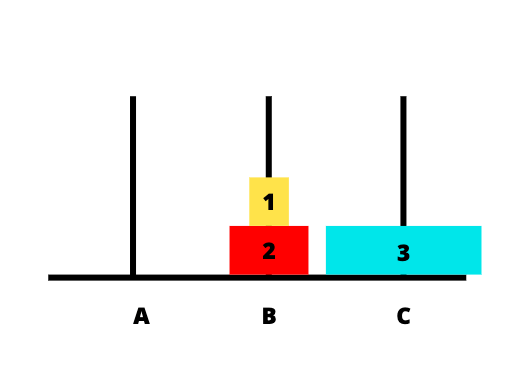
Di chuyển đĩa 2 từ cọc A sang cọc B



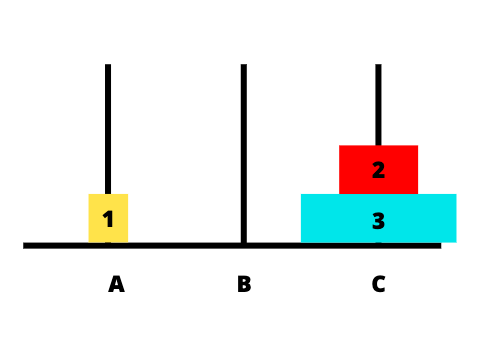
Di chuyển đĩa 1 từ cọc C sang trở lại cọc B



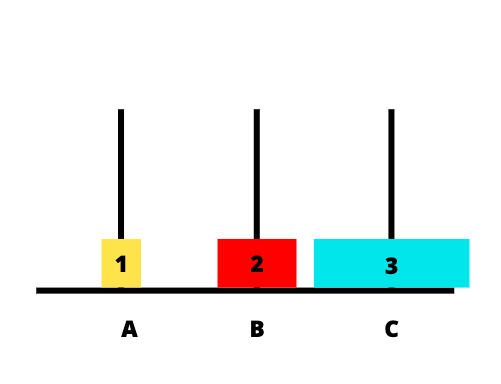
Di chuyển đĩa 3 từ cọc A sang cọc C



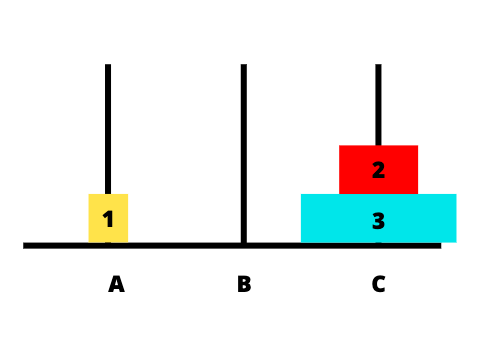
Di chuyển đĩa 2 từ cọc B sang cọc C



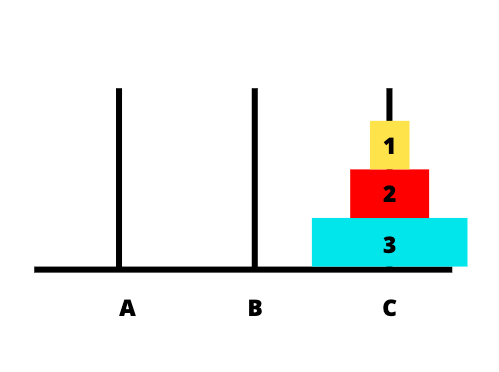
Move disk 1 from cọc B trở lại cọc A



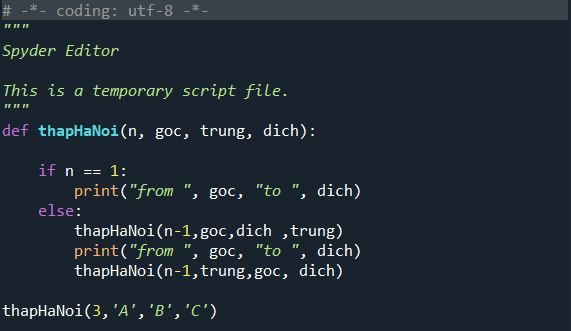
Di chuyển đĩa 2 từ cọc B sang cọc C



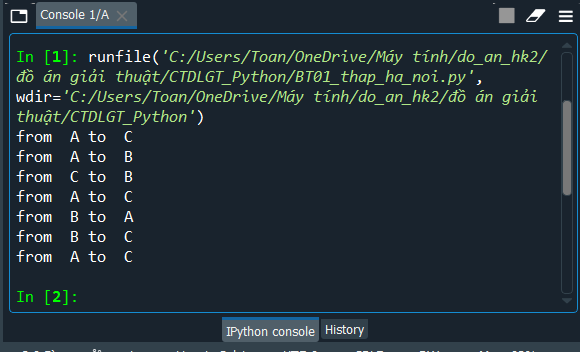
Move disk 1 from cọc A sang cọc C



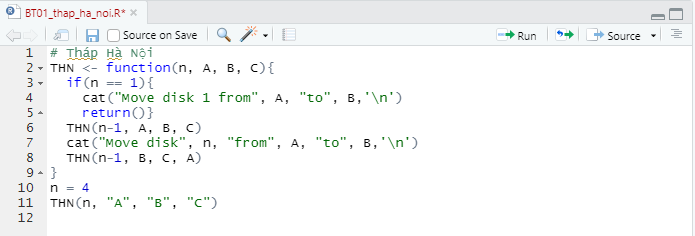
Code Python



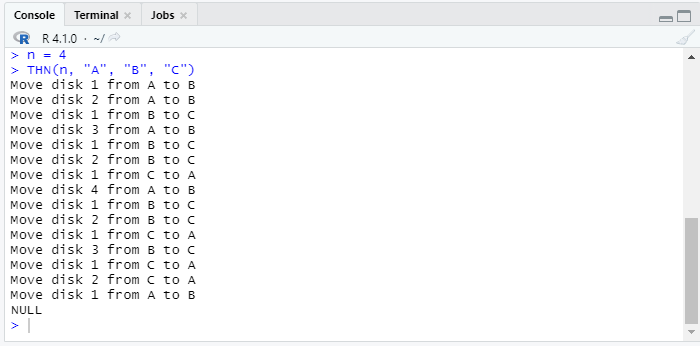
Kết quả



Code R



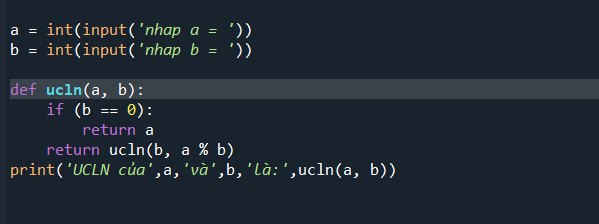
Kết quả



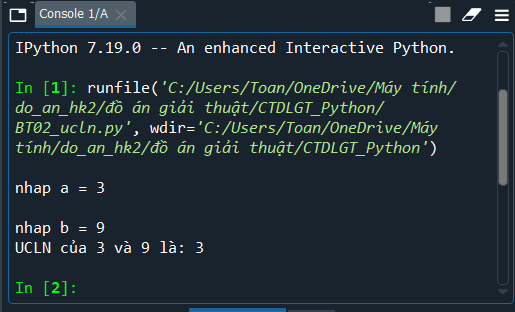
# ƯỚC SỐ CHUNG NHỎ NHẤT

Ước chung lớn nhất của hai hay nhiều số là số lớn nhất trong tập hợp ước chung. Để được chọn làm ước chung lớn nhất của hai số thì số đó phải là số lớn nhất trong tập hợp ước chung.

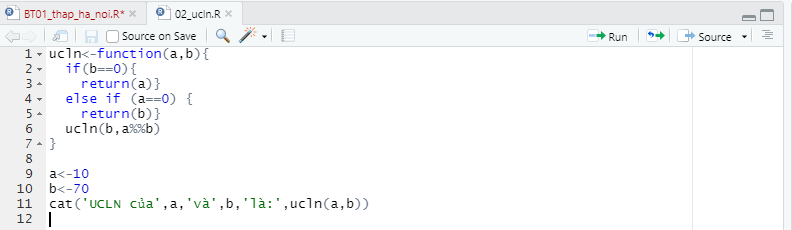
Code Python



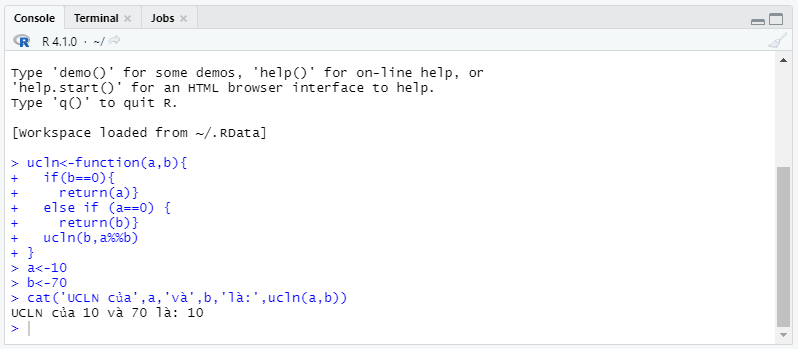
Kết quả



Code R



Kết quả



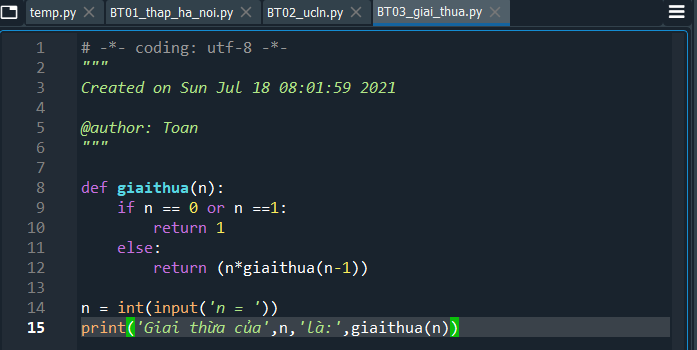
# TÍNH GIAI THỪA

Chương trình giai thừa trong ngôn ngữ lập trình : Ba phương pháp để tìm giai thừa, sử dụng vòng lặp for, sử dụng đệ quy và bằng cách tạo hàm. Như các bạn đã biết, trong toán học giai thừa được biểu diễn bằng cách sử dụng kí hiệu: '!'. Ví dụ vậy năm giai thừa sẽ được viết là (5!), N giai thừa là (n!). Ngoài ra, n! = n \* (n-1) \* (n-2) \* (n-3) ... 3.2.1 và giai thừa 0 được định nghĩa là một ví dụ, 0! = 1.

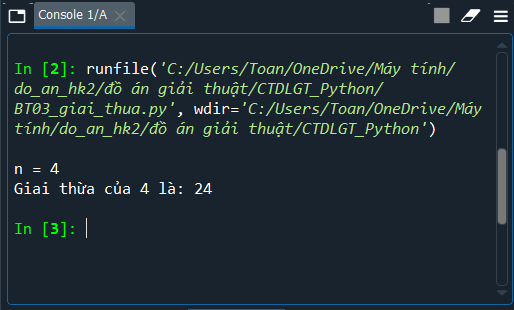
Giai thừa của một số 'n' là sản phẩm của tất cả các số từ 1 đến số 'n'

nó được ký hiệu bằng n !. Ví dụ n = 5 thì giai thừa 5 sẽ là 1 \* 2 \* 3 \* 4 \* 5 = 120. 5! = 120

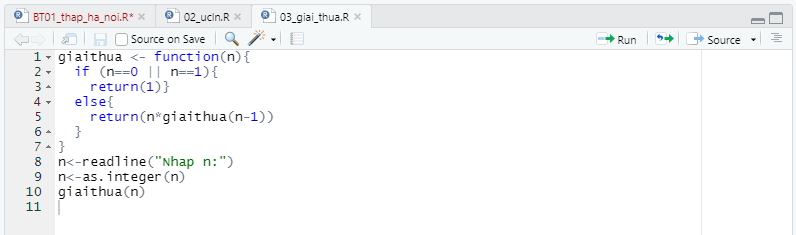
Code Python



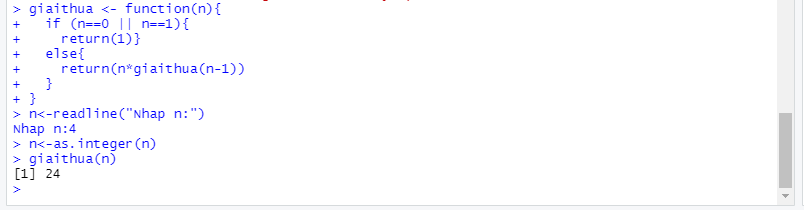
Kết quả



Code R



Kết quả



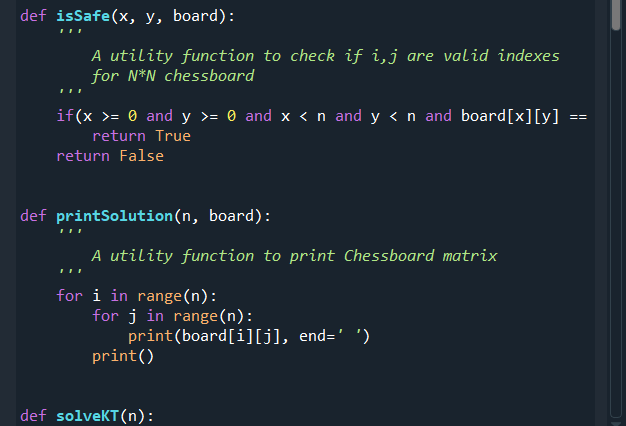
# MÃ ĐI TUẦN

Mã đi tuần (hay hành trình của quân mã) là bài toán về việc di chuyển một quân mã trên bàn cờ vua (8 x 8). Quân mã được đặt ở một ô trên một bàn cờ trống nó phải di chuyển theo quy tắc của cờ vua để đi qua mỗi ô trên bàn cờ đúng một lần.

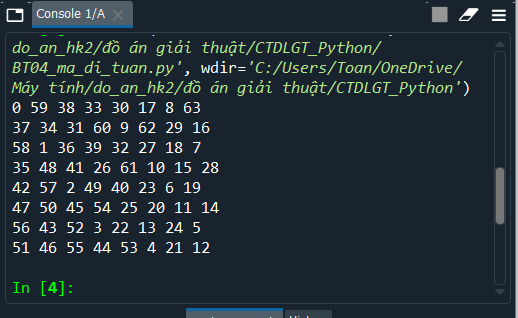
Nếu một quân mã đi hết 64 vị trí và tại vị trí cuối cùng có thể di chuyển đến vị trí bắt đầu thông qua một nước cờ thì đó gọi là một hành trình đóng

Có những hành trình, trong đó quân mã sau khi đi hết tất cả 64 ô của bàn cờ và từ ô cuối của hành trình không thể đi về ô xuất phát chỉ bằng một nước đi. Những hành trình như vậy được gọi là hành trình mở.

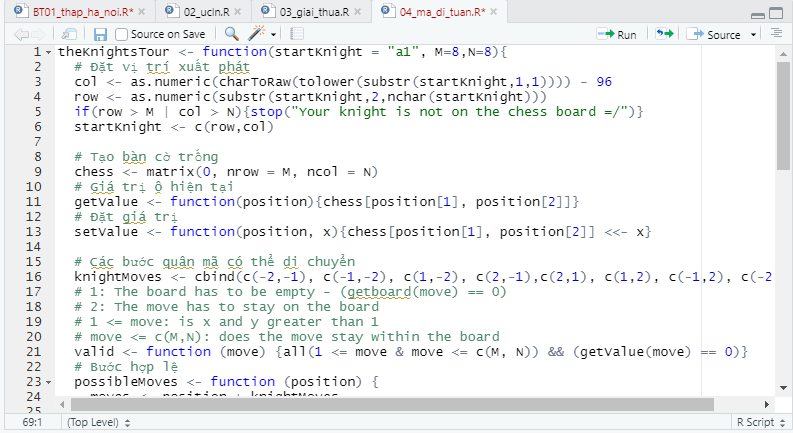
Code Python



Kết quả



Code R



Kết quả

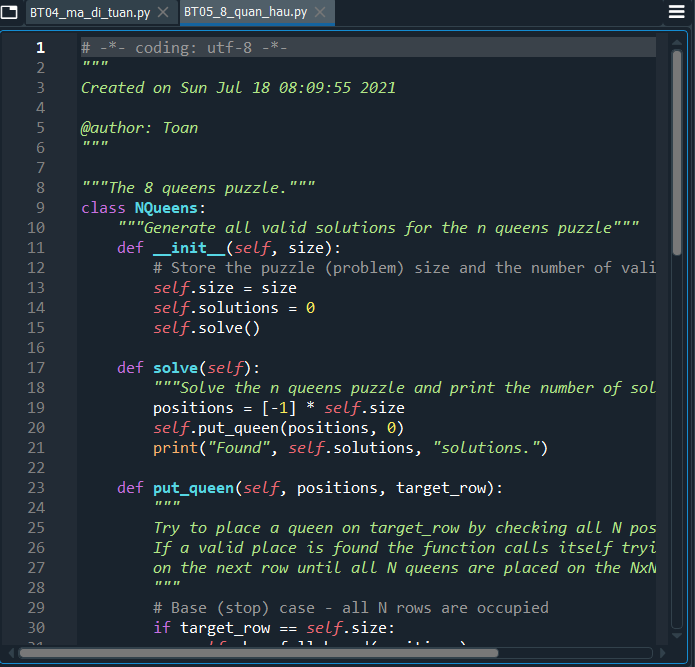


# 8 QUÂN HẬU

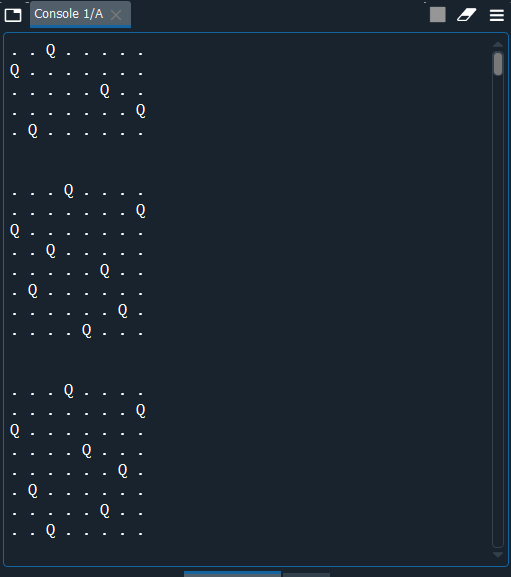
# Bài toán 8 con Hậu là bài toán nổi tiếng trong lĩnh vực toán học. Trong bài toán này thiết lập thường xuyên được nhắc nhở khi có liên quan đến chủ đề và đệ trình quay lui hoặc là trí tuệ nhân tạo.

Trên cờ có kích thước 8x8 ô, người chơi đặt các quân Hậu ở mỗi cột sao cho không có quân Hậu nào khác thường, nghĩa là không có quân Hậu nào có chung hàng hoặc cùng chung một đường với quân Hậu đã tồn tại trước đó.

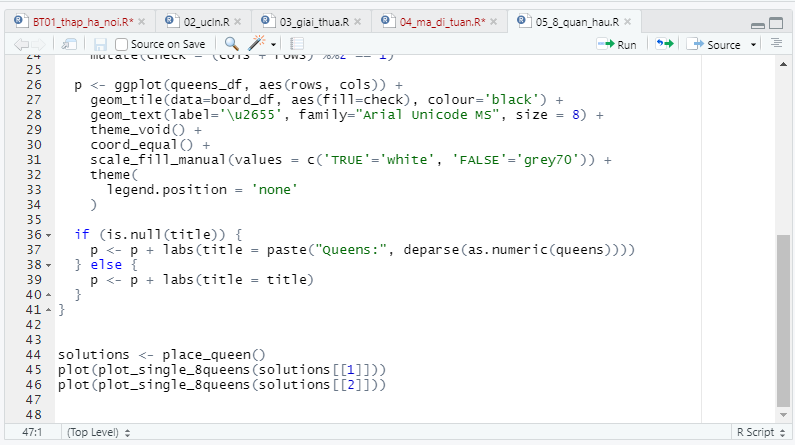
Code Python



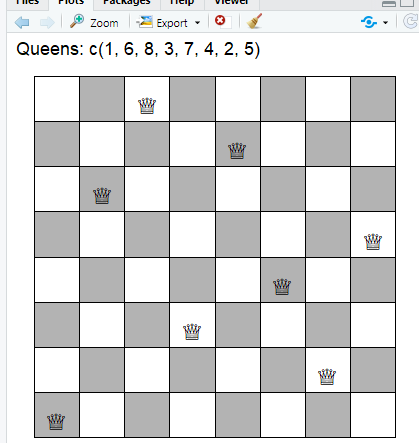
Kết quả



Code R



Kết quả



# DANH SÁCH LIÊN KẾT ĐƠN

Về bản chất, danh sách liên kết có chức năng như một mảng, có thể thêm và xóa các phần tử ở bất kỳ vị trí nào khi cần thiết. Một số sự khác nhau giữa danh sách liên kết và mảng:

|  |  |  |
| --- | --- | --- |
| **Nội dung** | **Mảng** | **Danh sách liên kết** |
| Kích thước | * Kích thước cố định * Cần chỉ rõ kích thước trong khi khai báo | * *Kích thước thay đổi trong quá trình thêm/ xóa phần tử* * *Kích thước tối đa phụ thuộc vào bộ nhớ* |
| Cấp phát bộ nhớ | * Tĩnh: Bộ nhớ được cấp phát trong quá trình biên dịch | * *Động: Bộ nhớ được cấp phát trong quá trình chạy* |
| Thứ tự & sắp xếp | * Được lưu trữ trên một dãy ô nhớ liên tục | * *Được lưu trữ trên các ô nhớ ngẫu nhiên* |
| Truy cập | * *Truy cập tới phần tử ngẫu nhiên trực tiếp bằng cách sử dụng chỉ số mảng: O(1)* | * Truy cập tới phần tử ngẫu nhiên cần phải duyệt từ đầu/cuối đến phần tử đó: O(n) |
| Tìm kiếm | * *Tìm kiếm tuyến tính hoặc tìm kiếm nhị phân* | * Chỉ có thể tìm kiếm tuyến tính |

Danh sách liên kết đơn là một tập hợp các Node được phân bố động, được sắp xếp theo cách sao cho mỗi Node chứa “*một giá trị”(Data)* và “*một con trỏ”(Next).*Con trỏ sẽ trỏ đến phần tử kế tiếp của danh sách liên kết đó. Nếu con trỏ mà trỏ tới NULL, nghĩa là đó là phần tử cuối cùng của linked list.

Danh sách các kiểu danh sách liên kết:

* Danh sách liên kết đơn(Single linked list): Chỉ có sự kết nối từ phần tử phía trước tới phần tử phía sau.
* Danh sách liên kết đôi(Double linked list): Có sự kết nối 2 chiều giữa phần tử phía trước với phần tử phía sau
* Danh sách liên kết vòng(Circular Linked List): Có thêm sự kết nối giữa 2 phần tử đầu tiên và phần tử cuối cùng để tạo thành vòng khép kín.
* **Các thao tác cơ bản**
  + Tạo danh sách rỗng
  + Thêm một phần tử vào danh sách
  + Duyệt danh sách
  + Tìm kiếm một giá trị trên danh sách
  + Xóa một phần tử ra khỏi danh sách
  + Hủy toàn bộ danh sách
  + …
* **Thêm một phần tử vào danh sách**: Có 3 vị trí thêm
  + Gắn vào đầu danh sách
  + Gắn vào cuối danh sách
  + Chèn vào sau nút q trong danh sách
* Chú ý trường hợp danh sách ban đầu rỗng

Danh sách liên kết đơn chủ yếu được dùng để xây dựng các loại cấu trúc dữ liệu khác như ngăn xếp hoặc hàng đợi hoặc đồ thị ...

Đôi khi ta cần biết node ở vị trí trước. Nhưng với danh sách liên kết đơn thì không làm được. Để biết được vị trí của cả các node trước và sau node hiện tại, ta phải lưu cả vị trí của node trước và node sau. Lúc này cấu trúc của một node sẽ có dạng như sau.

struct node {

int data;

struct node\* next;

struct node\* prev;

};

Cấu trúc trên chính là cấu trúc của danh sách liên kết kép.

# DANH SÁCH LIÊN KẾT KÉP

Danh sách liên kết kép là danh sách mà mỗi nút được nối với nhau theo hai

chiều. Mỗi nút là một cấu trúc gồm 3 trường:

- Trường thứ nhất lưu giá trị dữ liệu của nút đó – Infor

- Trường thứ hai là một con trỏ, trỏ đến phần tử kế tiếp - next

- Trường thứ ba là một con trỏ, trỏ đến phần tử liền trước – pre

Trong danh sách liên kết kép nút đầu tiên Head và nút cuối cùng là Tail. Để

duyệt danh sách liên kết kép ta có 2 cách: hoặc bắt đầu từ Head và dựa vào liên

kết next để đi đến phần tử cuối hoặc bắt đầu từ phần tử cuối Tail và dựa vào liên

kết pre để đi về phần tử đầu tiên.

! Mỗi phần tử trong danh sách kép là một cấu trúc gồm hai thành phần:

- thành phần dữ liệu: để lưu trữ hay mô tả thông tin được lưu trữ

trong từng nút của danh sách.

- thành phần liên kết: gồm hai thành phần để chỉ mối liên kết với

phần tử trước và phần tử sau trong danh sách.

* 1. Cơ chế

Thêm 1 nút vào ds

- Có 4 loại thao tác chèn new\_node vào danh sách:

* Cách 1: Chèn vào đầu danh sách
* Cách 2: Chèn vào cuối danh sách
* Cách 3 : Chèn vào danh sách sau một phần tử q
* Cách 4 : Chèn vào danh sách trước một phần tử q

Hủy phần tử

- Có 5 loại thao tác thông dụng hủy một phần tử ra khỏi

danh sách liên kết đôi:

*  Hủy phần tử đầu ds
*  Hủy phần tử cuối ds
*  Hủy một phần tử đứng sau phần tử q
*  Hủy một phần tử đứng trước phần tử q
*  Hủy 1 phần tử có khóa k

3.Ứng dụng của danh sách liên kết kép trong thực tế

Ta có thể ứng dụng danh sách liên kết kép trong khá nhiều trường hợp, ví dụ:

* Trong một phần mềm chơi nhạc có nút next và prev.
* Trong bộ đệm của trình duyệt web cho bạn quay trở lại trang trước hoặc tới trang kế tiếp.
* Các phần mềm sử dụng tính năng undo và redo ( MS word, các IDE).

# NGĂN XẾP STACK

Stack là một kiểu danh sách tuyến tính đặc biệt mà phép bổ sung và phép loại bỏ luôn luôn được thực hiện ở một đầu (gọi là đỉnh).

Hay còn có một định nghĩa khác: Ngăn xếp Stack là một cấu trúc dữ liệu trừu tượng làm việc theo nguyên lý vào sau ra trước LIFO (last in first out).

**Ví dụ Stack trong thực tế**: Khi chúng ta bỏ các cái bánh vào một chiếc hộp, hành động bỏ vào tương tự như Push trong Stack, vì chúng ta sẽ bỏ lần lượt từng cái bánh vào trên đỉnh của chiếc hộp. Và khi ta muốn lấy bánh ra ta cũng sẽ phải lấy cái ở trên đỉnh ra trước, hành động này tương tự như Pop trong Stack.

Một ngăn xếp là một cấu trúc dữ liệu dạng thùng chứa (container) của các phần tử (thường được gọi là các Node). Có hai thao tác cơ bản là ***push***và ***pop***.

* **Push**bổ sung một phần tử vào đỉnh (top) của ngăn xếp, nghĩa là nó sẽ được thêm vào sau các phần tử đã có sẵn trong ngăn xếp.
* **Pop**giải phóng và trả về phần tử đang đứng ở đỉnh của ngăn xếp. Phần tử sau khi được lấy sẽ bị xóa khỏi ngăn xếp.

Ngoài ra, Stack cũng có một số thao tác bổ trợ khác:

* **isEmpty()**: Kiểm tra xem stack có rỗng hay không.
* **Top()**: Trả về giá trị của phần tử nằm ở đầu Stack mà không hủy nó khỏi Stack. Nếu Stack rỗng thì thông báo và không thực hiện được thao tác này.

Trong phần này mình sẽ mô tả Stack bằng hai cách: mô tả Stack bằng mảng và mô tả Stack bằng danh sách liên kết đơn. Các bạn hãy xem hai cách này có gì giống và khác nhau nhé.

Mô tả Stack bằng mảng

Khi mô tả Stack bằng mảng, ta có một số đặc điểm sau:

* Việc bổ sung một phần tử vào Stack tương đương với việc thêm một phần tử ở cuối mảng.
* Việc loại bỏ một phần tử khỏi Stack tương đương với việc loại bỏ một phân tử ở cuối mảng.
* Stack sẽ bị tràn nếu bổ sung vào mảng đã đầy.
* Stack là rỗng khi số phần tử thực sự đang chưa trong mảng = 0.

Mô tả Stack bằng danh sách liên kết đơn

Khi mô tả Stack bằng danh sách liên kết đơn, ta cũng có một số đặc điểm sau:

Việc bổ sung một phần tử vào Stack tương đương với việc thêm một phần tử vào cuối danh sách (insertlast).

Việc loại bỏ một phần tử trong Stack cũng tương đương với việc loại bỏ một phần tử ở cuối danh sách.

Stack bị tràn khi vùng không gian nhớ dùng cho các biến động không còn đủ để thêm một phần tử mới. Tuy nhiên việc kiểm tra này rất khó bởi nó phụ thuộc vào máy tính và ngôn ngữ lập trình. Vì vậy khi cài đặt ta có thể bỏ qua việc kiểm tra Stack tràn.

Ứng dụng của Stack là gì?

* Chuyển đổi Infix to Postfix
* Tính năng undo(hoàn lại) ở nhiều nơi như chỉnh sửa, photoshop.
* Tính năng chuyển tiếp và lùi trong trình duyệt web
* Được sử dụng trong nhiều thuật toán như Tower of Hanoi, duyệt cây, bài toán nhịp cổ phiếu, bài toán biểu đồ.
* Các ứng dụng khác có thể là Backtracking, Knight tour problem, N queen problem và sudoku solver
* Trong các thuật toán đồ thị như Sắp xếp theo cấu trúc liên kết và các thành phần được kết nối mạnh mẽ

# HÀNG ĐỢI QUEUE

Hàng đợi (Queue) là một cấu trúc dữ liệu trừu tượng. Đặc điểm của hàng đợi là **FIFO (first in first out) - có nghĩa là vào trước ra trước**. Đặt tên là hàng đợi bởi vì nó là một cái gì đó tương tự như hàng đợi trong đời sống hàng ngày (xếp hàng).



Khác với ngăn xếp, hàng đợi là mở ở cả hai đầu. Một đầu luôn luôn được sử dụng để chèn dữ liệu vào (hay còn gọi là sắp vào hàng) và đầu kia được sử dụng để xóa dữ liệu (rời hàng). Cấu trúc dữ liệu hàng đợi tuân theo phương pháp First-In-First-Out, tức là dữ liệu được nhập vào đầu tiên sẽ được truy cập đầu tiên.

Trong đời sống thực chúng ta có rất nhiều ví dụ về hàng đợi, chẳng hạn như hàng xe ô tô trên đường một chiều (đặc biệt là khi tắc xe), trong đó xe nào vào đầu tiên sẽ thoát ra đầu tiên. Một vài ví dụ khác là xếp hàng học sinh, xếp hàng mua vé, …

Các hoạt động cơ bản trên cấu trúc dữ liệu hàng đợi

Các hoạt động trên cấu trúc dữ liệu hàng đợi có thể liên quan tới việc khởi tạo hàng đợi, sử dụng dữ liệu trên hàng đợi và sau đó là xóa dữ liệu khỏi bộ nhớ. Danh sách dưới đây là một số hoạt động cơ bản có thể thực hiện trên cấu trúc dữ liệu hàng đợi:

* **Hoạt động enqueue()**: thêm (hay lưu trữ) một phần tử vào trong hàng đợi.
* **Hoạt động dequeue()**: xóa một phần tử từ hàng đợi.

Để sử dụng hàng đợi một cách hiệu quả, chúng ta cũng cần kiểm tra trạng thái của hàng đợi. Để phục vụ cho mục đích này, dưới đây là một số tính năng hỗ trợ khác của hàng đợi:

* **Phương thức peek()**: lấy phần tử ở đầu hàng đợi, mà không xóa phần tử này.
* **Phương thức isFull()**: kiểm tra xem hàng đợi là đầy hay không.
* **Phương thức isEmpty()**: kiểm tra xem hàng đợi là trống hay hay không.

Trong cấu trúc dữ liệu hàng đợi, chúng ta luôn luôn: (1) dequeue (xóa) dữ liệu được trỏ bởi con trỏ **front** và (2) enqueue (nhập) dữ liệu vào trong hàng đợi bởi sự giúp đỡ của con trỏ **rear**.

Có nhiều bạn sẽ thắc mắc cái cấu trúc dữ liệu này ứng dụng trong các bài toán thế nào? Loại cấu trúc này sẽ áp dụng vào các bài toán cần có tính logic thứ tự. Ví dụ như việc xử lý các tiến trình của máy tính. Hoặc quản lý các hóa đơn đặt hàng . . .

# DUYỆT CÂY TRƯỚC ĐẾN SAU VÀ SAU ĐẾN TRƯỚC

Duyệt cây là một tiến trình để truy cập tất cả các nút của một cây và cũng có thể in các giá trị của các nút này. Bởi vì tất cả các nút được kết nối thông qua các cạnh (hoặc các link), nên chúng ta luôn luôn bắt đầu truy cập từ nút gốc. Do đó, chúng ta không thể truy cập ngẫu nhiên bất kỳ nút nào trong cây. Có ba phương thức mà chúng ta có thể sử dụng để duyệt một cây:

* Duyệt tiền thứ tự (Pre-order Traversal)
* Duyệt hậu thứ tự (Post-order Traversal)

Nói chung, chúng ta duyệt một cây để tìm kiếm hay là để xác định vị trí phần tử hoặc khóa đã cho trong cây hoặc là để in tất cả giá trị mà cây đó chứa

Cơ chế:

- Giải thuật cho cách duyệt tiền thứ tự ( duyệt cây trước đến sau)

Duyệt cho tới khi tất cả các nút đều được duyệt:

Bước 1: Truy cập nút gốc

Bước 2: Duyệt các cây con bên trái một cách đệ qui

Bước 3: Duyệt các cây con bên phải một cách đệ qui

- Giải thuật cho cách duyệt hậu thứ tự ( duyệt cây sau đến trước)

Duyệt cho tới khi tất cả các nút đều được duyệt:

Bước 1: Duyệt các cây con bên trái một cách đệ qui

Bước 2: Duyệt các cây con bên phải một cách đệ qui

Bước 3: Truy cập nút gốc.

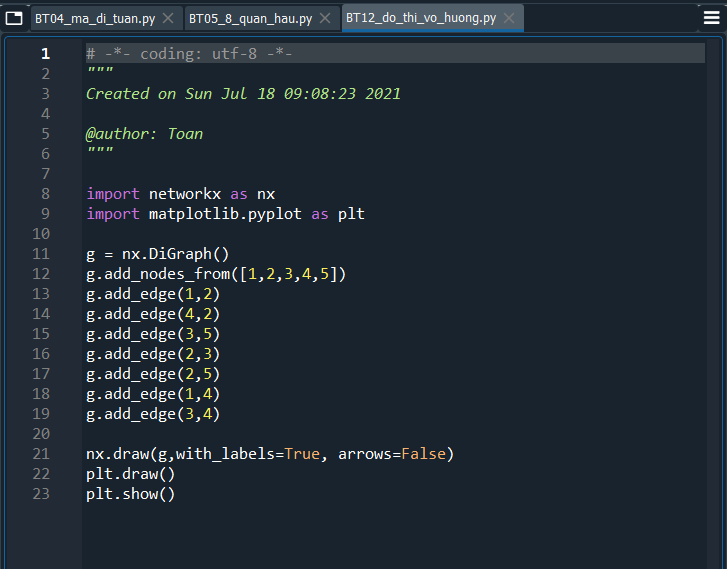
# 

# ĐỒ THỊ VÔ HƯỚNG

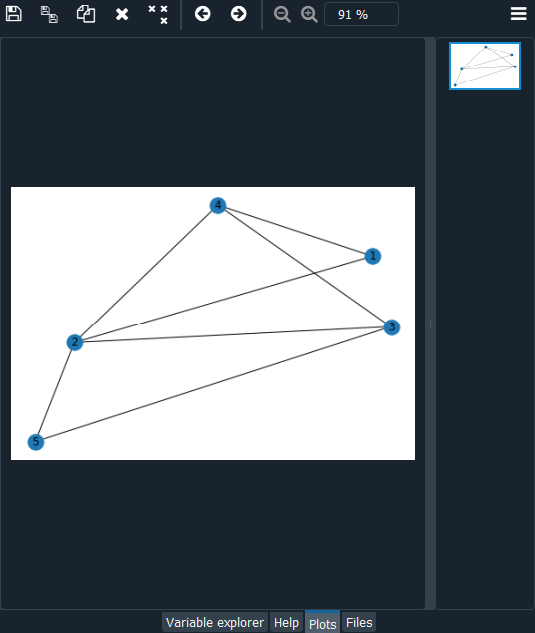
**Đồ thị vô hướng** hoặc **đồ thị** *G* là một [cặp không có thứ tự](https://vi.wikipedia.org/w/index.php?title=C%E1%BA%B7p_kh%C3%B4ng_c%C3%B3_th%E1%BB%A9_t%E1%BB%B1&action=edit&redlink=1) *(unordered pair)* *G*:=(*V*, *E*), trong đó

* *V*, [tập](https://vi.wikipedia.org/wiki/T%E1%BA%ADp_h%E1%BB%A3p) các **đỉnh** hoặc **nút**,
* *E*, tập các cặp không thứ tự chứa các đỉnh phân biệt, được gọi là **cạnh**. Hai đỉnh thuộc một cạnh được gọi là các **đỉnh** đầu cuối của cạnh đó.

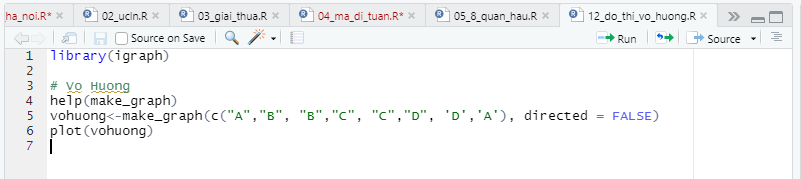
Trong nhiều tài liệu, tập các cạnh bao gồm cả các cặp đỉnh không phân biệt, các cạnh này được gọi là các khuyên. *V* (và *E*) thường là các tập hữu hạn, phần lớn các kết quả nghiên cứu đã biết không đúng (hoặc khác) khi áp dụng cho **đồ thị vô hạn** (infinite graph) vì nhiều luận cứ không dùng được trong trường hợp vô hạn.

Code Python

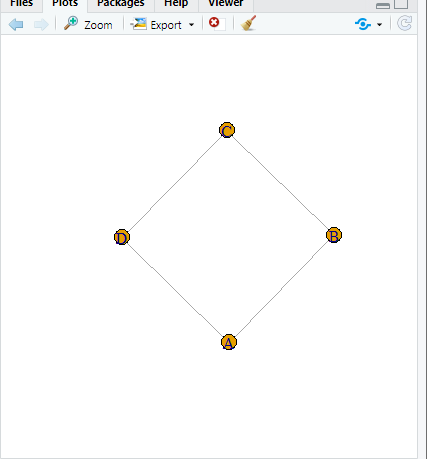
Kết quả



Code R



Kết quả

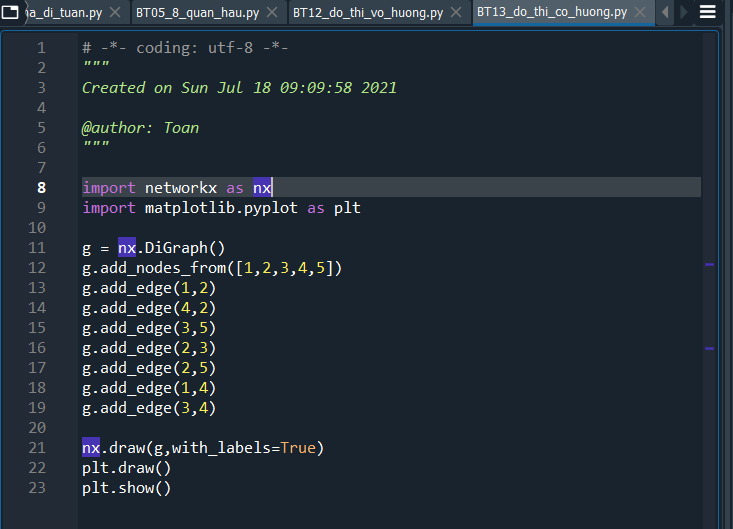


# ĐỒ THỊ CÓ HƯỚNG

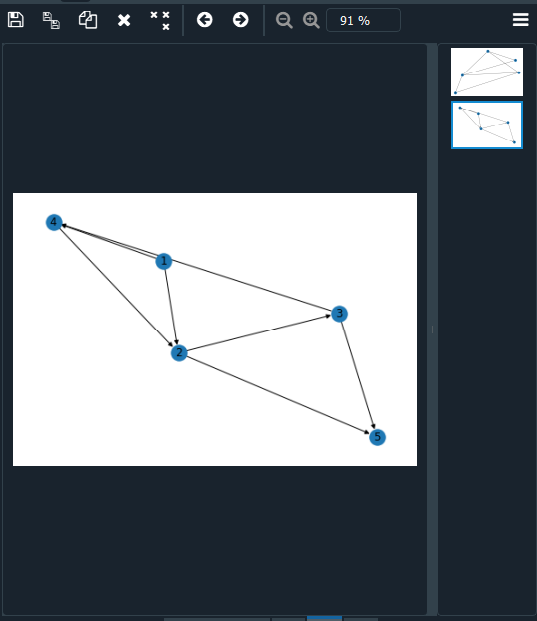
**Đồ thị có hướng** *G* là một cặp có thứ tự *G*:=(*V*, *A*), trong đó

* *V*, [tập](https://vi.wikipedia.org/wiki/T%E1%BA%ADp_h%E1%BB%A3p) các **đỉnh** hoặc **nút**,
* *A*, tập các cặp có thứ tự chứa các đỉnh, được gọi là các **cạnh có hướng** hoặc **cung**. Một cạnh *e* = (*x*, *y*) được coi là có hướng **từ** *x* **tới** *y*; *x* được gọi là **điểm đầu/gốc** và *y* được gọi là **điểm cuối/ngọn** của cạnh.

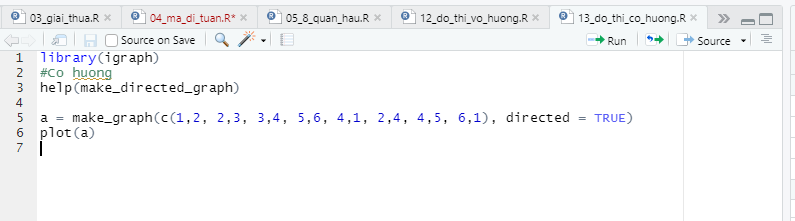
Code Python

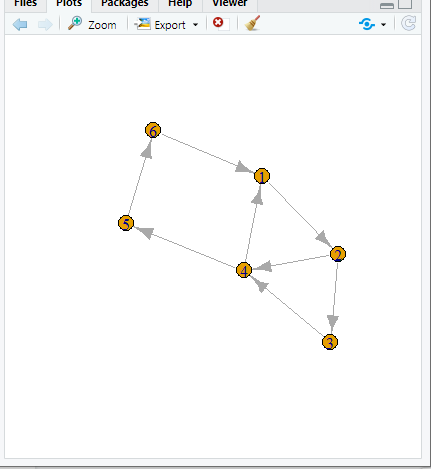


Kết quả



Code R



Kết quả

# SELECTION SORT

**Selection Sort** (**sắp xếp chọn**) là một **thuật toán sắp xếp đơn giản** dựa trên **so sánh tại chỗ**, trong đó:  
Danh sách được chia thành hai phần (Trái - Phải) (Vẫn là cùng một mảng nhé)

* Phần được sắp xếp ở đầu bên trái và phần chưa được sắp xếp ở đầu bên phải
* Lúc đầu thì phần bên phải là toàn bộ danh sách. (Vì phần bên trái chưa sắp xếp mà)
* Mỗi lần lặp chúng ta sẽ liên tục tìm giá trị nhỏ nhất ở phần bên phải, hoán đổi vị trí của nó cho phần tử ngoài cùng bên trái.

1. Cơ chế

* Giả sử, ta chọn được phần tử có giá trị nhỏ nhất nhất trên mảng là A[k]. với vị trí là k.
* Tráo đổi A[0] với A[k], vậy thì lúc này ta sẽ nhận được A[0] là phần tử có giá trị nhỏ nhất.
* Giả sử đến bước thứ i ta đã có A[0] <= A[1] <= ... <= A[i-1]. Bây giờ ta cần tìm thành phần có giá trị nhỏ nhất trong các phần tử từ A[i] đến A[n-1].
* Giả sử phần tử đó có vị trí là t có giá trị A[t] sao cho i <= t <= n - 1
* Ta lại tráo đổi A[i] với A[t]. Lặp lại cho tới i = n - 1
* Cuồi cùng, ta có mảng A được sắp xếp.

Thuật toán này không phù hợp với các tập dữ liệu lớn vì độ phức tạp trung bình.  
  
Khi bạn sắp xếp với một cơ sở dữ liệu lớn thì quá trình này sẽ chậm và tốn nhiều bộ nhớ máy tính.  
  
Độ phức tạp của selection sort là: O(n2)

Để bạn dần hiểu rõ hơn về thuật toán Selection Sort, hãy xem giải thuật của nó:

* Bước 1: Chọn phần tử có khóa nhỏ nhất trong n phần tử từ a[0] đến a[n-1] và hoán vị nó với phần tử a[0].
* Bước 2: Chọn phần tử có khóa nhỏ nhất trong n – 1 phần tử từ a[1] đến a[n-1] và hoán vị nó với a[1].
* Tổng quát ở bước thứ i chọn phần tử có khóa nhỏ nhất trong n – i phần tử từ a[i] đến a[n-1] và hoán vị nó với a[i].

Sau n – 1 bước thì mảng đã được sắp xếp.

Ứng dụng:

* Trong thực tế, chúng ta không cần tạo một danh sách mới cho các phần tử được sắp xếp, những gì chúng ta làm là coi phần ngoài cùng bên trái của danh sách là phân đoạn được sắp xếp.
* Sau đó chúng ta tìm kiếm phần tử nhỏ nhất trong toàn bộ danh sách đã cho và hoán đổi nó với phần tử đầu tiên.
* Thuật toán Selection Sort là một trong những thuật toán đơn giản nhất, dễ học và cũng dễ vận dụng vào các dự án thực tế.

# INSERTION SORT

Sắp xếp chèn là một thuật toán sắp xếp đơn giản hoạt động tương tự như cách bạn sắp xếp các thẻ chơi trong tay. Mảng hầu như được chia thành một phần được sắp xếp và một phần chưa được sắp xếp. Các giá trị từ phần chưa được sắp xếp được chọn và đặt ở vị trí chính xác trong phần được sắp xếp.

Cơ chế :

Để sắp xếp một mảng có kích thước n theo thứ tự tăng dần:

1: Lặp lại từ arr [1] đến arr [n] trên mảng.

2: So sánh phần tử hiện tại với phần tử trước của nó.

3: Nếu phần tử chính nhỏ hơn phần tử trước của nó, hãy so sánh nó với các phần tử trước đó. Di chuyển các phần tử lớn hơn lên một vị trí để tạo khoảng trống cho phần tử được hoán đổi.

* **Dưới đây là thuật toán sắp xếp chèn đơn giản cho danh sách liên kết.**

1) Tạo danh sách (hoặc kết quả) được sắp xếp trống

2) Duyệt qua danh sách đã cho, thực hiện theo các bước sau cho mọi nút.

…… a) Chèn nút hiện tại theo cách được sắp xếp trong danh sách đã sắp xếp hoặc kết quả.

3) Thay đổi phần đầu của danh sách liên kết đã cho thành phần đầu của danh sách được sắp xếp (hoặc kết quả).

Ứng dụng

* Thuật toán sử dụng trung bình n2/4 phép so sánh và n2/4 lần hoán vị, n2/2 phép so sánh và n2/2 lần hoán vị trong trường hợp xấu nhất, n-1 phép so sánh và 0 lần hoán vị trong trường hợp tốt nhất.
* Thuật toán thích hợp đối với mảng đã được sắp xếp một phần hoặc mảng có kích thước nhỏ.

# BUBBLE SORT

Sắp xếp nổi bọt (tiếng Anh: bubble sort) là một thuật toán sắp xếp đơn giản, với thao tác cơ bản là so sánh hai phần tử kề nhau, nếu chúng chưa đứng đúng thứ tự thì đổi chỗ (swap). Có thể tiến hành từ trên xuống (bên trái sang) hoặc từ dưới lên (bên phải sang). Sắp xếp nổi bọt còn có tên là sắp xếp bằng so sánh trực tiếp. Nó sử dụng phép so sánh các phần tử nên là một giải thuật sắp xếp kiểu so sánh.

Cơ chế:

Sắp xếp từ trên xuống

Giả sử dãy cần sắp xếp có n phần tử. Khi tiến hành từ trên xuống, ta so sánh hai phần tử đầu, nếu phần tử đứng trước lớn hơn phần tử đứng sau thì đổi chỗ chúng cho nhau. Tiếp tục làm như vậy với cặp phần tử thứ hai và thứ ba và tiếp tục cho đến cuối tập hợp dữ liệu, nghĩa là so sánh (và đổi chỗ nếu cần) phần tử thứ n-1 với phần tử thứ n. Sau bước này phần tử cuối cùng chính là phần tử lớn nhất của dãy.

Sau đó, quay lại so sánh (và đổi chố nếu cần) hai phần tử đầu cho đến khi gặp phần tử thứ n-2....

Ghi chú: Nếu trong một lần duyệt, không phải đổi chỗ bất cứ cặp phần tử nào thì danh sách đã được sắp xếp xong.

Sắp xếp từ dưới lên

Sắp xếp từ dưới lên so sánh (và đổi chỗ nếu cần) bắt đầu từ việc so sánh cặp phần tử thứ n-1 và n. Tiếp theo là so sánh cặp phần tử thứ n-2 và n-1,... cho đến khi so sánh và đổi chỗ cặp phần tử thứ nhất và thứ hai. Sau bước này phần tử nhỏ nhất đã được nổi lên vị trí trên cùng (nó giống như hình ảnh của các "bọt" khí nhẹ hơn được nổi lên trên). Tiếp theo tiến hành với các phần tử từ thứ 2 đến thứ n.

Ứng dụng

Do tính đơn giản của nó, sắp xếp nổi bọt thường được sử dụng để giới thiệu khái niệm về thuật toán sắp xếp.

Mặc dù là giải thuật khá chậm trong số các thuật toán sắp xếp nhưng thuật toán Bubble Sort vẫn luôn có chỗ đứng trong việc giải quyết các vấn đề thực tế.

Chẳng hạn như việc phát hiện một lỗi rất nhỏ (như hoán đổi chỉ hai phần tử) trong các mảng gần như được sắp xếp và sửa nó chỉ với độ phức tạp tuyến tính (2n) trong đồ hoạ máy tính.

# QUICK SORT

Sắp xếp nhanh (Quicksort), còn được gọi là sắp xếp kiểu phân chia (part sort) là một thuật toán sắp xếp phát triển bởi C.A.R. Hoarec sắp thành hai danh sách con. Khác với sắp xếp trộn, chia danh sách cần sắp xếp {\displaystyle a[1..n]}{\displaystyle a[1..n]} thành hai danh sách con có kích thước tương đối bằng nhau nhờ chỉ số đứng giữa danh sách, sắp xếp nhanh chia nó thành hai danh sách bằng cách so sánh từng phần tử của danh sách với một phần tử được chọn được gọi là phần tử chốt. Những phần tử nhỏ hơn hoặc bằng phần tử chốt được đưa về phía trước và nằm trong danh sách con thứ nhất, các phần tử lớn hơn chốt được đưa về phía sau và thuộc danh sách đứng sau. Cứ tiếp tục chia như vậy tới khi các danh sách con đều có độ dài bằng 1.

Cơ chế:

* + Phần tử chốt (pivot)

Kỹ thuật chọn phần tử chốt ảnh hưởng khá nhiều đến khả năng rơi vào các vòng lặp vô hạn đối với các trường hợp đặc biệt. Tốt nhất là chọn phần tử chốt là trung vị của danh sách. Khi đó sau log2(n) lần phân chia ta sẽ đạt tới kích thước danh sách bằng 1. Tuy nhiên điều đó rất khó. Có các cách chọn phần tử chốt như sau:

Chọn phần tử đứng đầu hoặc đứng cuối làm phần tử chốt.

Chọn phần tử đứng giữa danh sách làm phần tử chốt.

Chọn phần tử trung vị trong 3 phần tử đứng đầu, đứng giữa và đứng cuối làm phần tử chốt.

Chọn phần tử ngẫu nhiên làm phần tử chốt. (Cách này có thể dẫn đến khả năng rơi vào các trường hợp đặc biệt)

* + Thuật phân chia

Sau khi phần tử chốt được chọn giải thuật phân chia nên tiến hành như thế nào?

Một giải pháp đơn giản nhất cho vấn đề này là duyệt từ đầu đến cuối lần lượt so sánh các phần tử của danh sách với phần tử chốt. Theo cách này, ta phải tiến hành n phép so sánh, ngoài ra còn phải dành n đơn vị bộ nhớ để lưu giữ các giá trị trung gian.

Một giải pháp khác được đề nghị là duyệt theo hai đường. Một đường từ đầu danh sách, một đường từ cuối danh sách. Theo cách này, ta tìm phần tử đầu tiên tính từ trái lớn hơn phần tử chốt và phần tử đầu tiên phía phải nhỏ hơn hoặc bằng phần tử chốt rồi đổi chỗ cho nhau. Tiếp tục như vậy cho đến khi hai đường gặp nhau.

Để có thể gọi đệ quy ta xét bài toán phân chia một danh sách con của a: a[k1,k2 ] thành hai danh sách.

Ứng dụng

* Mấu chốt chính của thuật toán quick sort là việc phân đoạn dãy số (Xem hàm partition()). Mục tiêu của công việc này là: Cho một mảng và một phần tử x là pivot. Đặt x vào đúng vị trí của mảng đã sắp xếp. Di chuyển tất cả các phần tử của mảng mà nhỏ hơn x sang bên trái vị trí của x, và di chuyển tất cả các phần tử của mảng mà lớn hơn x sang bên phải vị trí của x.
* Khi đó ta sẽ có 2 mảng con: mảng bên trai của x và mảng bên phải của x. Tiếp tục công việc với mỗi mảng con(chọn pivot, phân đoạn) cho tới khi mảng được sắp xếp.

# HEAP SORT

Thuật toán Heap sort là một kỹ thuật sắp xếp dựa trên so sánh dựa trên cấu trúc dữ liệu Binary Heap.

Nó tương tự như sắp xếp lựa chọn, nơi đầu tiên chúng ta tìm phần tử lớn nhất và đặt phần tử lớn nhất ở cuối.

Chúng ta lặp lại quá trình tương tự cho các phần tử còn lại.

Một Binary Heap là một cây nhị phân hoàn chỉnh trong đó các mục được lưu trữ theo một thứ tự đặc biệt sao cho giá trị trong nút cha lớn hơn (hoặc nhỏ hơn) so với giá trị trong hai nút con của nó.

Loại trước được gọi là max heap và loại sau được gọi là min heap.

Heap có thể được biểu diễn bằng một cây hoặc mảng nhị phân.

Cơ chế:

* Thuật toán Heap sort lấy ý tưởng giải quyết từ cấu trúc heap, cụ thể:
* Ta coi dãy cần sắp xếp là một cây nhị phân hoàn chỉnh, sau đó hiệu chỉnh cây thành dạng cấu trúc heap ( vun đống)
* Dựa vào tính chất của cấu trúc heap, ta có thể lấy được ra phần từ lớn nhất hoặc nhỏ nhất của dãy, phần tử này chính là gốc của heap. Giảm số lượng phần tử của cây nhị phân và tái cấu trúc heap.
* Đưa phần tử đỉnh heap về đúng vị trí của dãy ở cuối mảng, sau đó giảm số lượng phần tử của mảng (không xét tới phần tử cuối nữa)
* Tái cấu trúc heap và lặp lại việc lấy phần tử gốc của cấu trúc heap cho tới khi danh sách ban đầu chỉ còn 1 phần tử. Đưa phần tử này về đúng vị trí và kết thúc thuật toán.
* Ta phải thực hiện tái cấu trúc heap, vun lại đống bởi vì sau khi lấy ra phần tử gốc heap, cấu trúc heap không còn nữa.

Ứng dụng:

* Ngoài giải thuật sắp xếp vun đống, cấu trúc đống còn được ứng dụng trong nhiều giải thuật khác, khi cần lấy ra nhanh chóng các phần tử lớn nhất (hoặc nhỏ nhất) của một dãy phần tử, chẳng hạn trong hàng đợi có ưu tiên trong đó tiêu chuẩn ưu tiên là có khóa lớn nhất (hoặc nhỏ nhất). Có thể tìm thấy điều đó trong giải thuật tìm bộ mã Huffman cho một bảng tần số của các ký tự.
* Có độ phức tạp trung bình O (n log n) trong mọi trường hợp. Là một trong các thuật toán sắp xếp nhanh nhất
* Ít bị ảnh hưởng bởi dữ liệu đầu vào, có thể ứng dụng nhiều trong thực tế.

# MERGE SORT

Trong khoa học máy tính, sắp xếp trộn (merge sort) là một thuật toán sắp xếp để sắp xếp các danh sách (hoặc bất kỳ cấu trúc dữ liệu nào có thể truy cập tuần tự, v.d. luồng tập tin) theo một trật tự nào đó. Nó được xếp vào thể loại sắp xếp so sánh. Thuật toán này là một ví dụ tương đối điển hình của lối thuật toán chia để trị do John von Neumann đưa ra lần đầu năm 1945[1]. Một thuật toán chi tiết được Goldstine và Neumann đưa ra năm 1948.

Cơ chế:

* Giả sử có hai danh sách đã được sắp xếp a[1..m] và b[1..n.]. Ta có thể trộn chúng lại thành một danh sách mới c[1..m+n] được sắp xếp theo cách sau:
* So sánh hai phần tử đứng đầu của hai danh sách, lấy phần tử nhỏ hơn cho vào danh sách mới. Tiếp tục như vậy cho tới khi một trong hai danh sách là rỗng.
* Khi một trong hai danh sách là rỗng ta lấy phần còn lại của danh sách kia cho vào cuối danh sách mới.
* rộn tại chỗ
* Giả sử trong danh sách a[1..n] có 2 danh sách con kề nhau a[k\_{1}..k\_{2}] và a[k\_{2}+1..k\_{3}] đã được sắp. Ta áp dụng cách trộn tương tự như trên để trộn hai danh sách con vào một danh sách tạm T[k\_{1}..k\_{3}] rồi trả lại các giá trị của danh sách tạm T về danh sách A. Làm như vậy gọi là trộn tại chỗ.
* Sắp xếp trộn đệ quy
* Một cách gọi đệ quy của sắp xếp trộn cũng thường được hướng dẫn trong các giáo trình giải thuật.
* Để sắp xếp trộn đoạn a[k\_{1}..k\_{2}] của danh sách a[1..n] ta chia đoạn đó thành 2 phần a[k\_{1}..k\_{3}] và a[k\_{3}+1..k\_{2}],trong đó k\_{3}=int((k\_{1}+k\_{2})/2) tiến hành sắp xếp với mỗi phần rồi trộn chúng lại. Lời gọi thủ tục sắp xếp trộn với a[1..n] sẽ cho kết quả là sắp toàn bộ danh sách a[1..n]
* Trộn từ dưới lên
* Nếu danh sách con chỉ gồm hai phần tử, mỗi nửa của nó gồm một phần tử đương nhiên đã được sắp. Do đó việc trộn tại chố hai nửa danh sách này cho danh sách con 2 phân tử được sắp.
* Xuất phát từ đầu danh sách a ta trộn a[1] với a[2], a[3] với a[4],... Khi đó mọi danh sách con gồm hai phần tử của a đã được sắp. Tiếp tục trộn các danh sách con kế tiếp nhau gồm 2 phần tử thành các danh sách con 4 phần tử... Mỗi lần trộn số các danh sách con cần trộn giảm đi một nửa. Quá trình dừng lại khi số danh sách con chỉ còn một.
* Trộn các đường tự nhiên
* Như trong phần đánh giá giải thuật, một trong những nhược điểm lớn của thuật toán Trộn trực tiếp là không tận dụng những thông tin về đặc tính của dãy cần sắp xếp. Ví dụ trường hợp dãy đã có thứ tự sẵn. Chính vì vậy, trong thực tế người ta ít dùng thuật toán trộn trực tiếp mà người ta dùng phiên bản cải tiến của nó. Phiên bản này thường được biết với tên gọi thuật toán trộn tự nhiên (Natural Merge sort).

Ứng dụng:

* Merge Sort rất hữu ích để sắp xếp các danh sách được liên kết trong thời gian O (nLogn). Trong trường hợp danh sách được liên kết, trường hợp này khác nhau chủ yếu do sự khác biệt trong phân bổ bộ nhớ của mảng và danh sách được liên kết. Không giống như mảng, các nút danh sách liên kết có thể không liền kề trong bộ nhớ. Không giống như một mảng, trong danh sách liên kết, chúng ta có thể chèn các mục vào giữa trong O (1) không gian thừa và O (1) thời gian. Do đó hoạt động hợp nhất của sắp xếp hợp nhất có thể được thực hiện mà không có thêm dung lượng cho danh sách được liên kết. Trong mảng, chúng ta có thể thực hiện truy cập ngẫu nhiên khi các phần tử nằm kề nhau trong bộ nhớ. Giả sử chúng ta có một mảng A số nguyên (4 byte) và đặt địa chỉ của A [0] là x thì để truy cập A [i], chúng ta có thể truy cập trực tiếp vào bộ nhớ tại (x + i \* 4). Không giống như mảng, chúng ta không thể thực hiện truy cập ngẫu nhiên trong danh sách liên kết. Sắp xếp nhanh yêu cầu rất nhiều loại truy cập. Trong danh sách liên kết để truy cập chỉ mục thứ i, chúng ta phải di chuyển từng nút từ đầu đến nút thứ i vì chúng ta không có khối bộ nhớ liên tục. Do đó, chi phí tăng đối với nhanh chóng. Sắp xếp hợp nhất truy cập dữ liệu một cách tuần tự và nhu cầu truy cập ngẫu nhiên thấp.
* Vấn đề đảo ngược số lượng
* Được sử dụng trong sắp xếp bên ngoài