ĐẠI HỌC HUẾ



**KHOA KỸ THUẬT VÀ CÔNG NGHỆ**

🙠🙟🕮🙝🙢

Icon

Description automatically generated

**BÁO CÁO ĐỒ ÁN**

**HỌC KÌ I, năm học 2022-2023**

**Học phần:**

**CẤU TRÚC DỮ LIỆU & GIẢI THUẬT**

**Số phách**

**(**Do hội đồng chấm ghi thi)

***Thừa Thiên Huế, tháng 04 năm 2023.***

ĐẠI HỌC HUẾ



**KHOA KỸ THUẬT VÀ CÔNG NGHỆ**

🙠🙟🕮🙝🙢

Icon

Description automatically generated

**BÁO CÁO ĐỒ ÁN**

**HỌC KÌ I, năm học 2022-2023**

**Học phần:**

**CẤU TRÚC DỮ LIỆU & GIẢI THUẬT**

**Giảng viên hướng dẫn:** Nguyễn Thanh Nam.

**Lớp:** Khoa học dữ liệu và Trí tuệ nhân tạo khóa 3.

**Sinh viên thực hiện:** Phạm Phước Bảo Tín.

(ký và ghi rõ họ tên)

**Số phách**

**(**Do hội đồng chấm ghi thi)

***Thừa Thiên Huế, tháng 04 năm 2023.***

# LỜI CẢM ƠN

Được trở thành sinh viên Khoa Kỹ Thuật và Công Nghệ - Đại học Huế em rất hạnh phúc và biết ơn. Hạnh phúc vì mình đã đạt được mục tiêu mong muốn và biết ơn sự cống hiến, chỉ bảo tận tình sâu sắc của quý thầy cô trong khoa đồng thời đã tạo điều kiện học tập lí tưởng cho chúng em. Để hoàn thành đồ án một cách chỉnh chu nhất có thể em xin gửi lời cảm ơn đến thầy giáo bộ môn – Thầy Nguyễn Thanh Nam đã hướng dẫn tận tình, chi tiết cho chúng em trong quá trình hoàn thành đồ án lẫn quá trình học tập học kì đầu tiên của đời sinh viên chúng em. Hi vọng rằng thời gian sắp tới em sẽ luôn cố gắng, nổ lực hơn nữa trong học tập chuyên nghành của mình.

Trong quá trình hoàn thành đồ án mặc dù em đã chuẩn bị kĩ nhưng không thể tránh khỏi những sai sót, em mong nhận được sự góp ý từ quý thầy, cô. Lời cuối cùng em xin kính chúc quý thầy, cô thật nhiều sức khỏe để tiếp tục dẫn dắt chúng em và những thế hệ tiếp theo thành người.

# DANH MỤC HÌNH ẢNH

[Hình 1: Mô tả thuật toán đệ quy Tháp Hà Nội 5](#_Toc132656901)

# DANH MỤC BẢNG BIỂU

[Bảng 1: Mã nguồn bài toán Tháp Hà Nội 6](#_Toc133225291)

[Bảng 2: Mã nguồn bài toán tìm USCLN bằng đệ quy 6](#_Toc133225292)

[Bảng 3: Mã nguồn bài toán tìm USCLN bằng vòng lặp 6](#_Toc133225293)

[Bảng 4: Mã nguồn bài toán mã đi tuần 8](#_Toc133225294)

[Bảng 5: Mã nguồn bài toán tám quân hậu 9](#_Toc133225295)

[Bảng 6: Mã nguồn danh sách liên kết đơn 11](#_Toc133225296)

# MỤC LỤC

[LỜI CẢM ƠN i](#_Toc133257440)

[DANH MỤC HÌNH ẢNH ii](#_Toc133257441)

[DANH MỤC BẢNG BIỂU iii](#_Toc133257442)

[MỤC LỤC iv](#_Toc133257443)

[CHƯƠNG 1: ĐỆ QUY 5](#_Toc133257444)

[1.1 Tháp Hà Nội 5](#_Toc133257445)

[1.1.1 Giới thiệu về bài toán Tháp Hà Nội 5](#_Toc133257446)

[1.1.2 Giải thuật đệ quy trong bài toán Tháp Hà Nội 5](#_Toc133257447)

[1.1.3 Mã nguồn giải thuật đệ quy Tháp Hà Nội 5](#_Toc133257448)

[1.2 Bài toán tìm ước số chung lớn nhất 6](#_Toc133257449)

[1.2.1 Giới thiệu về bài toán tìm ước số chung lớn nhất của hai số 6](#_Toc133257450)

[1.2.1 Giải quyết bài toán USCLN bằng đệ quy 6](#_Toc133257451)

[1.2.2 Giải quyết bài toán USCL bằng vòng lặp đơn giản 6](#_Toc133257452)

[1.3 Bài toán mã đi tuần 7](#_Toc133257453)

[1.3.1 Giới thiệu bài toán mã đi tuần 7](#_Toc133257454)

[1.3.2 Mã nguồn bài toán mã đi tuần 7](#_Toc133257455)

[1.4 Bài toán tám quân hậu 8](#_Toc133257456)

[1.4.1 Giới thiệu bài toán tám quân hậu 8](#_Toc133257457)

[1.4.2 Mã nguồn bài toán tám quân hậu 8](#_Toc133257458)

[CHƯƠNG 2: DANH SÁCH LIÊN KẾT 10](#_Toc133257459)

[2.1 Danh sách liên kết đơn 10](#_Toc133257460)

[2.1.1 Giới thiệu danh sách liên kết đơn 10](#_Toc133257461)

[2.1.2 Cài đặt danh sách liên kết đơn 10](#_Toc133257462)

# CHƯƠNG 1: ĐỆ QUY

## Tháp Hà Nội

### Giới thiệu về bài toán Tháp Hà Nội

Bài toán Tháp Hà Nội là một trò chơi toán học. Yêu cầu của trò chơi là di chuyển toàn bộ số đĩa sang một cọc khác, tuân theo các quy tắc sau:

* + 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 đặt lên một đĩa lớn hơn nó.

Để giải bài toán này chúng ta có thể sử dụng giải thuật đệ quy.

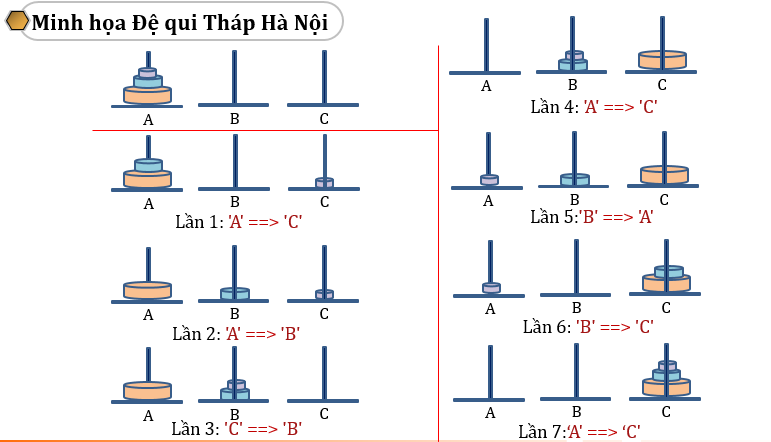
### Giải thuật đệ quy trong bài toán Tháp Hà Nội

Đặt tên cọc nguồn, trung gian, đích lần lượt là cọc A, B, C, gọi n là số đĩa cần di chuyển. Đánh số đĩa từ 1( đĩa nhỏ nhất, trên cùng) đến n ( đĩa lớn nhất, cuối cùng)

Để di chuyển n đĩa từ cột A sang cột C thì cần:

1. Chuyển (n-1) đĩa từ cọc A sang cọc B. Chỉ còn đĩa n ở cọc A.
2. Chuyển đĩa n từ cọc A sang cột C.
3. Chuyển đĩa (n-1) còn lại từ cọc B sang cọc C cho chúng nằm trên đĩa n.
4. Giải thuật không còn là đệ quy khi chỉ còn 1 đĩa, vì ta chỉ cần chuyển đĩa còn lại cuối cùng qua cọc đích mà không phải thông qua cọc trung gian.

Giải thuật được mô tả với số đĩa bằng 3 được minh họa như Hình 1.



Hình : Mô tả thuật toán đệ quy Tháp Hà Nội

### Mã nguồn giải thuật đệ quy Tháp Hà Nội

Dưới đây là mã nguồn giải bài toán Tháp Hà Nội bằng mã lệnh Python

|  |  |
| --- | --- |
| def thap\_ha\_noi(n,cot\_a,cot\_b,cot\_c):      if n==1:          print(f'Chuyển đĩa {n} từ {cot\_a} sang {cot\_c}  ')          return      thap\_ha\_noi(n-1,cot\_a,cot\_c,cot\_b)      print(f'Chuyển đĩa {n} từ { cot\_a} sang {cot\_c} ')      thap\_ha\_noi(n-1,cot\_b,cot\_a,cot\_c)  n=int(input('Nhap so dia:\t'))  thap\_ha\_noi(n,"a","b","c") | - Tham số n là số đĩa cần di chuyển, cot\_a, cot\_b, cot\_c lần lượt là cột nguồn, trung gian, đích.   * Nếu mà chỉ có một đĩa thì chuyển từ cột nguồn sang cột đích. |

Bảng : Mã nguồn bài toán Tháp Hà Nội

## Bài toán tìm ước số chung lớn nhất

### 1.2.1 Giới thiệu về bài toán tìm ước số chung lớn nhất của hai số

Ước số chung lớn nhất của hai số a và b là k, điều kiện a và b đều chia hết cho k và k lớn nhất.

Để giải quyết bài toán này chúng ta có thể sử dụng nhiều cách làm. Em xin trình bày hai cách gồm: sử dụng đệ quy và sử dụng vòng lặp thông thường.

### 1.2.1 Giải quyết bài toán USCLN bằng đệ quy

|  |  |
| --- | --- |
| def ucln1(a, b):      if a == 0 or b == 0:          return a if b == 0 else b      else:          return ucln(b, a % b) | - Đầu tiên kiểm tra hai số có bằng 0 hay khác 0, nếu cả hai số đều bằng 0 thì trả về None.  - Nếu cả hai số đều khác 0 thì gọi hàm chính nó với tham số b và số dư của phép chia a chia b. |

Bảng : Mã nguồn bài toán tìm USCLN bằng đệ quy

### 1.2.2 Giải quyết bài toán USCL bằng vòng lặp đơn giản

Để giải quyết bài toán tìm ước số chung lớn nhất của hai số, ngoài cách sử dụng đệ quy chúng ta còn có thể sử dụng vòng lặp đơn giản như dưới đây.

Trong bài toán này em sử dụng vòng lặp for để thực hiện.

|  |  |
| --- | --- |
| def ucln(m,n):      a=min(m,n)      for i in rang(a,0,-1):          if m%i==0 and n%i==0:              return i | - Sử dụng hàm có sẵn trong Python để tìm ra số bé hơn là a, vòng lặp for với biến i chạy từ số a về 1 với bước nhảy -1.  - Khi m và n đều chia hết cho 1 số thì trả về kết quả và thoát khỏi vòng lặp. |

Bảng : Mã nguồn bài toán tìm USCLN bằng vòng lặp

## Bài toán mã đi tuần

### 1.3.1 Giới thiệu bài toán mã đi tuần

Mã đi tuần hay hành trình của quân mã là bài toán về việc chuyển một quân mã trên bàn cờ vua. 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.

Nước đi của một quân mã giống hình chữ L và nó có thể di chuyển tất cả các hướng. Ở một vị trí thích hợp thì quân mã có thể di chuyển đến được 8 vị trí.

### 1.3.2 Mã nguồn bài toán mã đi tuần

Dưới đây là mã nguồn giải bài toán mã đi tuần như Bảng 4.

|  |  |
| --- | --- |
| def kiem\_tra\_nuoc\_di(x, y, board, n):      return x>=0 and x<n and y>=0 and y<n and board[x][y]==-1 | - Hàm kiem\_tra\_nuoc\_di kiểm tra nước đi mới di chuyển có hợp lệ hay không, hợp lí thì trả về True và ngược lại. |
| def ma\_di\_chuyen(board, x, y, moves, movei, n):      if movei == n\*n:          return True      for i in range(8):          next\_x = x+ moves[i][0]          next\_y = y+moves[i][1]      if kiem\_tra\_nuoc\_di(next\_x, next\_y, board, n)==True:              board[next\_x][next\_y] = movei              if ma\_di\_chuyen(board, next\_x, next\_y, moves, movei+1, n)==True:                  return True              board[next\_x][next\_y] = -1      return False | - Tạo hàm ma\_di\_chuyen để tìm đường đi cho quân mã  - Nếu số bước di chuyển của quân mã mà bằng số ô trên bàn cờ thì dừng lại.  - Lặp qua tất cả các các bước có thể đi của quân mã, next\_x và next\_y lần lượt là tọa độ trên bàn cờ.  - Sử dụng hàm kiem\_tra\_nuoc\_di nếu đúng thì bước đi ở ô đó đánh số thứ tự của bước đi.  - Thực hiện đệ quy hàm ma\_di\_chuyen như trước đó |
| def in\_ket\_qua(board):      for i in range(len(board)):          for j in range(len(board)):              print(board[i][j], end = ' ')          print() | - Hàm in\_ket\_qua có nhiệm vụ in ra bài toán đã giải quyết. |
| def kiem\_tra\_ket\_qua(n):      board = [[1 for i in range(n)] for j in range(n)]      moves = [(2,1),(1,2),(-1,2),(-2,1),(-2,-1),(-1,-2),(1,-2),(2,-1)]      board[0][0] = 0      if ma\_di\_chuyen(board, 0, 0, moves, 1, n)==False:          print("Không tìm thấy giải pháp")      else:          in\_ket\_qua(board) | - Hàm kiem\_tra\_ket\_qua , tạo ma bàn cờ mô hình với các vị trí trên bàn cờ với giá trị bằng -1.  - Gán vị trí bắt đầu với giá trị bằng 0  - moves là list gồm các nước mà quân mã có thể di chuyển trên bàn cờ.  - Nếu hàm ma\_di\_chuyen trả về False thì không có giải pháp cho bài toán và ngược lại thì gọi hàm in\_ket\_qua. |

Bảng : Mã nguồn bài toán mã đi tuần

## Bài toán tám quân hậu

## 1.4.1 Giới thiệu bài toán tám quân hậu

Bài toán yêu cầu đặt tám quân hậu trên bàn cờ để làm sao không có quân hậu nào có thể bị tấn công. Các nước đi của quân hậu hoàn toàn như trong cờ vua.

### 1.4.2 Mã nguồn bài toán tám quân hậu

Dưới đây là mã nguồn giải bài toán tám quân hậu như Bảng 5.

|  |  |
| --- | --- |
| def kiem\_tra(board, row, col):      for i in range(col):          if board[row][i] == 1:              return False      i, j = row, col      while i >= 0 and j >= 0:          if board[i][j] == 1:              return False          i -= 1          j -= 1      i, j = row, col      while i < len(board) and j >= 0:          if board[i][j] == 1:              return False          i += 1          j -= 1        return True | - Kiểm tra hàng ngang, nếu trong hàng đã có đặt quân hậu rồi thì trả về False.  - Mã lệnh phía bên sử dụng vòng lặp kiểm tra quân hậu đang tìm vị trí có bị tấn công theo chiều phía trên bên trái, nếu bị tấn công tiếp tục thử ô khác trong cột.  - Mã lệnh phía bên sử dụng vòng lặp để kiểm tra quân hậu đang tìm vị trí có bị tấn công theo chiều phía dưới bên trái, nếu bị tấn công trả về false và quay lại tiếp tục thử các ô khác trong cột.  - Nếu vị trí thỏa mãn thì đặt quân hậu vào vị trí đó |
| def tim\_kiem(board, col):      if col == len(board):          return True      for row in range(len(board)):          if kiem\_tra(board, row, col)==True:              board[row][col] = 1              if tim\_kiem(board, col+1)==True:                  return True              board[row][col] = 0      return False | - Hàm tim\_kiem được sử dụng để tìm kiếm vị trí đặt các quân hậu  - Nếu đã đặt được quân hậu vào cột cuối cùng thành công thì kết thúc bài toán.  - Sử dụng vòng lặp để kiểm tra vị trí mới có thỏa mãn hay không.  - Nếu vị trí mới thỏa mãn thì đệ quy đến cột tiếp theo.  - Nếu không tìm thấy giải pháp thì quay lui tìm giải pháp mới.  - Nếu không tìm được giải pháp thì trả về kết quả. |

Bảng : Mã nguồn bài toán tám quân hậu

# CHƯƠNG 2: DANH SÁCH LIÊN KẾT

## Danh sách liên kết đơn

### 2.1.1 Giới thiệu danh sách liên kết đơn

Danh sách liên kết đơn(Single linked list): Chỉ có sự kết nối từ phần tử phía trước tới phần tử phía sau. **Đây cũng là** ví dụ tốt nhất và đơn giản nhất về cấu trúc dữ liệu động sử dụng con trỏ để cài đặt.

### 2.1.2 Cài đặt danh sách liên kết đơn

Dưới đây là mã nguồn cài đặt danh sách liên kết đơn như Bảng 6.

|  |  |
| --- | --- |
| class Node:      def \_\_init\_\_(self, data):          self.data = data          self.next = None | - Lớp Node được tạo ra để đại diện cho một nút trong danh sách liên kết.  - Gồm hai thuộc tính: data để lưu dữ liệu và next để lưu trữ tham chiếu đến nút kế tiếp trong danh sách liên kết. |
| class Dslk:      def \_\_init\_\_(self):          self.dau=None          self.duoi=None | - Lớp Dslk để tạo và quản lí một danh sách liên kết, gồm nhiều phương thức như : tìm kiếm, chèn, xóa,...  - Phương thức khởi tạo gồm 2 thuộc tính: self.dau dùng để tham chiếu đến nút đầu tiên, self.duoi dùng để tham chiếu đến nút cuối cùng trong danh sách liên kết. |
| def add\_LinkedList(self, data):          node=Node(data)          if self.dau is None:              self.dau=node              self.duoi=node          else:              self.duoi.next=node              self.duoi=node | - Phương thức add\_LinkedList dùng để thêm 1 nút vào danh sách liên kế.  - Nếu nút đầu là rỗng thì nút được thêm vào sẽ là nút đầu tiên.  - Ngược lại ta thêm nút mới vào cuối danh sách bằng cách gán ‘next’ của nút cuối cùng danh sách hiện tại bằng nút mới, sau đó cập nhật nút cuối bằng nút mới thêm vào |
| def insert\_LinkedList(self,index,value):          node=Node(value)          before=None          now=self.dau          i=0          while i<index and now is not None:              i+=1              before=now              now=now.next          if before==None:              node.next=self.dau              self.dau=node              if self.duoi==None:                  self.duoi=node          else:              if now== None:                  self.duoi.next=node                  self.duoi=node              else:                  before.next=node                  node.next=now | - Phương thức chèn giá trị vào danh sách liên kết  - Biến before gán bằng rỗng, biến now tham chiếu đến nút đầu tiên trong danh sách liên kết.  - Sử dụng vòng lặp để tham chiếu đến nút vị thứ cần chèn, sau khi duyệt hết.  - Nếu before rỗng có nghĩa là nút đầu tiên rỗng nên sẽ chèn vào đầu danh sách.  - Ngược lại, now đang trỏ đến vị trí kế tiếp mà rỗng thì có nghĩa chèn vào cuối danh sách. Ngược lại now không rỗng thì chèn vào giữa danh sách. |

Bảng : Mã nguồn danh sách liên kết đơn

## 2.2 Danh sách liên kết đôi

## 2.2.1 Giới thiệu danh sách liên kết đôi

Danh sách liên kết đôi (hay còn gọi là danh sách liên kết kép) là một cấu trúc dữ liệu được sử dụng trong lập trình để lưu trữ và quản lý một danh sách các phần tử, trong đó mỗi phần tử bao gồm hai phần: một giá trị và hai con trỏ, mỗi con trỏ trỏ tới phần tử phía trước và phía sau của phần tử đó trong danh sách.

### 2.2.2 Cài đặt danh sách liên kết đôi

### 2.3 Ngăn xếp (Stack)

### 2.3.1 Giới thiệu ngăn xếp –(Stack)

Ngăn xếp (stack) là một cấu trúc dữ liệu trừu tượng, được sử dụng để lưu trữ một tập hợp các phần tử. Các phần tử được lưu trữ theo cách xếp chồng lên nhau, trong đó phần tử cuối cùng được thêm vào (còn gọi là đỉnh của stack) là phần tử đầu tiên được lấy ra.

### 2.3.2 Cài đặt ngăn xếp – (Stack)

Dưới đây là mã nguồn cài đặt ngăn xếp – (Stack) như Bảng 7.

|  |  |
| --- | --- |
| class Stack:      def \_\_init\_\_(self):          self.items=[]        def push(self,item):          self.items.append(item)        def \_\_len\_\_(self):          return (f'Độ dài ngăn sắp xếp: {len(self.items)}')      def pop(self):          if len(self.items==0): return " stack is empty"          return self.items.pop()      def stack\_is\_empty(self):          if len(self.items)==0:              return "Ngăn xếp rỗng"          else: return "stack is not empty "      def \_\_str\_\_(self):          return str(self.items) | - Tạo lớp Stack  - Phương thức khởi tạo danh sách liên kết rỗng.  - Phương thức push dùng để thêm phần tử vào danh sách liên kết.  - Mã lệnh phía bên trả về độ dài của ngăn xếp.  - Phương thức pop lấy ra phần tử cuối cùng của ngăn xếp, nếu ngăn xếp rỗng thì thông báo ngăn xếp rỗng.  - Phương thức stack\_is\_empty kiểm tra ngăn xếp có phải là rỗng hay không.  - Mã lệnh phía bên dùng để xuất ra màn hình các phần tử trong ngăn xếp. |

Bảng : Mã nguồn cài đặt ngăn xếp-(Stack)

## 2.4 Hàng đợi (Queue)

### 2.4.1 Giới thiệu hàng đợi (Queue)

Hàng đợi (Queue) là một cấu trúc dữ liệu dùng để chứa các đối tượng theo cơ chế FIFO ( First In First Out) có nghĩa là vào trước ra sau.

Trong hàng đợi, các đối tượng được thêm vào bất cứ lúc nào nhưng khi lấy ra chỉ được phép lấy ra phần tử đầu tiên trong hàng đợi. Việc thêm đối tượng vào hàng đợi luôn diễn ra ở cuối hàng đợi, ngược lại việc lấy đối tượng ra khỏi hàng đợi luôn diễn ra ở đầu hàng đợi.

### 2.4.2 Cài đặt hàng đợi (Queue)

Dưới đây là mã nguồn cài đặt hàng đợi ( Queue) như

|  |  |
| --- | --- |
| class Queue:      def \_\_init\_\_(self):          self.items = []        def is\_empty(self):          return len(self.items) == 0        def enqueue(self, item):          self.items.append(item)        def dequeue(self):          if self.is\_empty():              return None          return self.items.pop(0)        def \_\_len\_\_(self):          return len(self.items) | - Tạo lớp Queue - Phương thức khởi tạo hàng đợi rỗng.  - Mã lệnh phía bên là phương thức kiểm tra hàng đợi có rỗng hay không, trả về đúng hoặc sai.  - Phương thức enqueue thêm phần tử vào cuối hàng đợi.  - Phương thức dequeue có vai trò lấy phần tử đầu tiên trong hàng đợi theo quy tắc FIFO ( vào trước ra sau), nếu hàng đợi rồng thì trả về None.  - Phương thức len dùng để truy xuất độ dài hàng đợi hay còn gọi là số phần tử trong hàng đợi. |

Bảng : Mã nguồn cài đặt hàng đợi( Queue)