ĐẠI HỌC HUẾ
KHOA KỸ THUẬT VÀ CÔNG NGHỆ

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

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

DANH MỤC BẢNG BIỂU

MỤC LỤC

CHƯƠNG 1: ĐỆ QUY

1.1 Tháp Hà Nội

1.1.1 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.

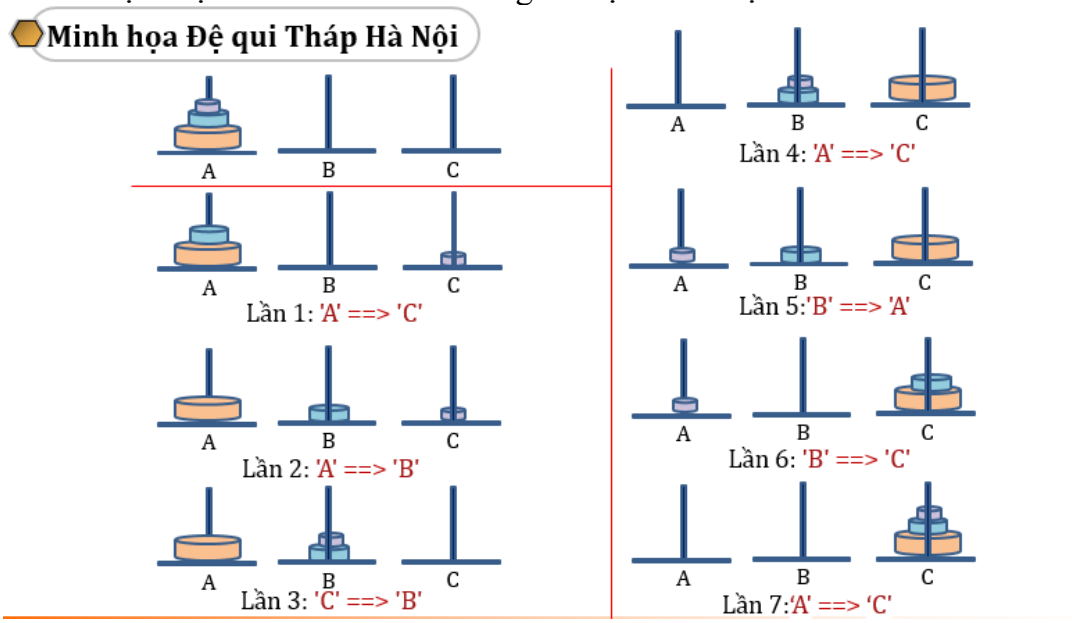
1.1.2 Giải thuật đệ quy trong bài toán Tháp Hà Nội

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 1: Mô tả thuật toán đệ quy Tháp Hà Nội

1.1.3 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 1: Mã nguồn bài toán Tháp Hà Nội

1.2 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 2: 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 3: Mã nguồn bài toán tìm USCLN bằng vòng lặp

1.3 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
	- 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

if ma_di_chuyen(board, 0, 0, moves, 1, n)==False:
  print("Không tìm thấy giải pháp")
else:
  in_ket_qua(board)
	- 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 4: Mã nguồn bài toán mã đi tuần

1.4 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
	- Kiểm tra hàng ngang, nếu trong hàng đã có đặt quân hậu rồi thì trả về False.

i, j = row, col
while i >= 0 and j >= 0:
  if board[i][j] == 1:
  return False
  i -= 1
  j -= 1
	- 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.

i, j = row, col
while i < len(board) and j >= 0:
  if board[i][j] == 1:
  return False
  i += 1
  j -= 1
 	
return True
	- 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 5: Mã nguồn bài toán tám quân hậu

CHƯƠNG 2: DANH SÁCH LIÊN KẾT

2.1 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
	- 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.

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
	- 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 6: 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()
	- 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.

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)
	- 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 7: 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 rỗng 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 8: Mã nguồn cài đặt hàng đợi(Queue)

CHƯƠNG 3: CÂY

3.1 Giới thiệu cấu trúc dữ liệu cây

Cây lập trình (Programming Tree) là một cấu trúc dữ liệu mô tả cách mà các khối mã (code blocks) được tổ chức trong một chương trình hoặc hàm. Các khối mã được đặt trong các nút của cây và quan hệ giữa chúng được mô tả bởi các cạnh của cây. Cây lập trình thường được sử dụng để biểu diễn cấu trúc chương trình hoặc hàm một cách rõ ràng, giúp cho việc đọc và hiểu mã nguồn trở nên dễ dàng hơn.

Các loại cây lập trình phổ biến bao gồm cây cú pháp (Syntax Tree), cây thực thi (Execution Tree) và cây AST (Abstract Syntax Tree). Cây cú pháp biểu diễn cấu trúc cú pháp của chương trình, cây thực thi mô tả các bước thực thi của chương trình và cây AST biểu diễn cấu trúc trừu tượng của chương trình một cách rõ ràng và đơn giản hơn.

3.1.1 Ý nghĩa cấu trúc dữ liệu cây

Việc sử dụng cây lập trình mang lại nhiều lợi ích trong việc lập trình và phát triển phần mềm. Dưới đây là một số ý nghĩa của việc sử dụng cây lập trình:

1. Biểu diễn cấu trúc chương trình một cách rõ ràng: Cây lập trình giúp biểu diễn cấu trúc của chương trình một cách rõ ràng và dễ hiểu hơn. Các khối mã được tổ chức theo một thứ tự nhất định, giúp cho việc đọc và hiểu mã nguồn trở nên dễ dàng hơn.
2. Dễ dàng tìm lỗi: Cây lập trình cũng giúp cho việc tìm lỗi trong chương trình trở nên dễ dàng hơn. Với cây lập trình, các lỗi có thể được xác định và giải quyết nhanh chóng, giúp tiết kiệm thời gian và công sức trong việc sửa lỗi.
3. Phát triển phần mềm dễ dàng hơn: Sử dụng cây lập trình giúp cho việc phát triển phần mềm trở nên dễ dàng hơn. Việc biểu diễn cấu trúc của chương trình một cách rõ ràng giúp cho các nhà phát triển dễ dàng hơn trong việc tạo, thay đổi và cải tiến chương trình.
4. Dễ dàng đọc và hiểu mã nguồn: Cây lập trình giúp cho mã nguồn trở nên dễ đọc và hiểu hơn. Với cấu trúc rõ ràng, các nhà phát triển có thể dễ dàng hình dung được cấu trúc của chương trình mà không cần phải đọc toàn bộ mã nguồn.
5. Tăng tính cấu trúc và tái sử dụng của chương trình: Sử dụng cây lập trình giúp tăng tính cấu trúc và tái sử dụng của chương trình. Các khối mã được tổ chức một cách rõ ràng, giúp cho việc sử dụng lại mã nguồn trở nên dễ dàng hơn, từ đó giúp tiết kiệm thời gian và công sức trong quá trình phát triển phần mềm.

3.1.2 Cách thức hoạt động cấu trúc dữ liệu cây

Cây lập trình hoạt động bằng cách sử dụng các nút và liên kết giữa chúng để biểu diễn cấu trúc của một chương trình máy tính. Mỗi nút trong cây lập trình đại diện cho một thực thể trong chương trình, ví dụ như một toán hạng, một biến, một phép tính, hoặc một lệnh điều khiển. Các liên kết giữa các nút thể hiện mối quan hệ logic giữa chúng.

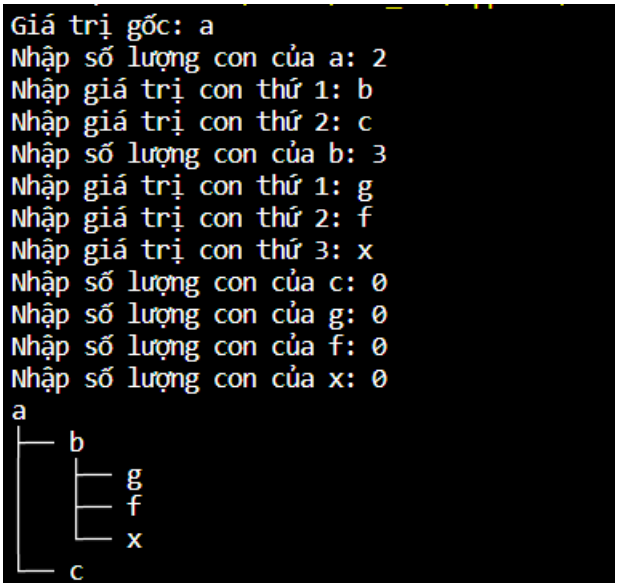
Cây lập trình thường được xây dựng dựa trên ngôn ngữ lập trình và có thể được tạo ra bằng tay hoặc bằng các công cụ tự động. Cây lập trình được sử dụng trong nhiều môi

trường lập trình khác nhau, bao gồm các trình biên dịch, trình thông dịch, trình gỡ lỗi, và các công cụ phân tích chương trình.

Khi một chương trình được thực thi, cây lập trình được duyệt theo từ trên xuống dưới bằng cách sử dụng các thuật toán duyệt cây như duyệt theo chiều sâu (depth-first) hoặc duyệt theo chiều rộng (breadth-first). Khi duyệt cây, các nút sẽ được thực thi tuần tự theo thứ tự được quy định bởi cấu trúc của cây lập trình.

Mã nguồn cài đặt cây trong ngôn ngữ python

```
1. import anytree as cay
2. from anytree import Node, RenderTree, NodeMixin
3. 
4. class Node:
5.     def __init__(self, ten):
6.         self.ten = ten
7.         self.children = []
8. 
9.     def them_cay(self, node):
10.         self.children.append(node)
11. 
12.def tao_cay():
13.    root_value = input("Giá trị gốc: ")
14.    root = Node(root_value)
15. 
16.    node_queue = [root]
17.    while len(node_queue) > 0:
18.        current_node = node_queue.pop(0)
19.        num_children = int(input(f"Nhập số lượng con của
   {current_node.ten}: "))
20.        for i in range(num_children):
21.            child_value = input(f"Nhập giá trị con thứ {i+1}: ")
22.            child_node = Node(child_value)
23.            current_node.them_cay(child_node)
24.            node_queue.append(child_node)
25. 
26.    return root
27.goc = tao_cay()
28.for pre, _, node in RenderTree(goc):
29.    print("%s%s" % (pre, node.ten))
```



BÀI 2: CÀI ĐẶT CÂY- DUYỆT CÂY THEO THỨ TỰ TRƯỚC

Nội dung

Duyệt cây theo thứ tự trước (pre-order traversal) là một cách duyệt qua tất cả các nút trong cây theo thứ tự nhất định. Khi duyệt cây theo thứ tự trước, trước tiên ta sẽ duyệt qua nút gốc, sau đó duyệt qua tất cả các nút con bên trái của nút gốc, và cuối cùng là tất cả các nút con bên phải của nút gốc.

Ý nghĩa

Việc sử dụng cài đặt cây và duyệt cây theo thứ tự trước là rất hữu ích trong việc giải quyết nhiều bài toán lập trình. Các ứng dụng của cây và duyệt cây theo thứ tự trước có thể bao gồm:

Tìm kiếm: Cây được sử dụng để lưu trữ và tìm kiếm các phần tử dựa trên giá trị. Khi duyệt cây theo thứ tự trước, ta có thể duyệt qua tất cả các nút trong cây và kiểm tra giá trị của chúng để tìm kiếm phần tử cần tìm.

CHƯƠNG 4: ĐỒ THỊ

4.1 Đồ thị vô hướng

4.2 Đồ thị có hướng

CHƯƠNG 5: SẮP XẾP

5.1 Sắp xếp chọn

5.1.1 Giới thiệu về sắp xếp chọn

Sắp xếp chọn là một thuật toán sắp xếp đơn giản, dựa trên việc so sánh tại chỗ. trong đó danh sách được chia thành hai phần, phần được sắp xếp (sorted list**)** ở bên trái và phần chưa được sắp xếp **(**unsorted list) ở bên phải. Ban đầu, phần được sắp xếp là trống và phần chưa được sắp xếp là toàn bộ danh sách ban đầu.

5.1.2 Cài đặt sắp xếp chọn

Dưới đây là mã nguồn sắp xếp chọn (tăng dần) như bảng Bảng 7.

def sap_xep_chon(lst):
  lst_1=lst.copy()
  for i in range(len(lst)-1):
  min=i
  for j in range(i+1,len(lst_1)):
  if lst_1[j]<lst_1[min]:
  min=j
  lst_1[min],lst_1[i]=lst_1[i],lst_1[min]
  return lst_1
	- Dựng hàm sap_xep_chon với tham số truyền vào là 1 list số thực.
	- Tạo một bản sao của list đó để có thể tái sử dụng.
	- Sử dụng vòng lặp lồng vòng lặp, gán biến min=i ở vị trí đầu tiền,sau đó sử dụng vòng lặp bắt đầu từ vị trí thứ i+1 đến phần tử cuối cùng, nếu gặp phần tử bé hơn vị trí min thì gán lại min bằng vị trí đó.
	- Sau khi ra khỏi vòng lặp thì hoán đổi vị trí min ban đầu với vị trí min vừa tìm thấy (lúc này trị trí có giá trị min sẽ lên đầu danh sách)

Bảng 9: Mã nguồn sắp xếp chọn

5.2 Sắp xếp chèn

5.2.1 Giới thiệu sắp xếp chèn

Sắp xếp chèn là một giải thuật sắp xếp dựa trên so sánh in-place. Ở đây, một danh sách con luôn luôn được duy trì dưới dạng đã qua sắp xếp. Sắp xếp chèn là chèn thêm một phần tử vào danh sách con đã qua sắp xếp. Phần tử được chèn vào vị trí thích hợp sao cho vẫn đảm bảo rằng danh sách con đó vẫn sắp theo thứ tự.

5.2.2 Cài đặt sắp xếp chèn

Dưới đây là mã nguồn sắp xếp chèn (tăng dần) như Bảng 10.

def sap_xep_chen(lst):
  lst_1=lst.copy()
  for i in range(len(lst)):
  index_min=i
  min=lst_1[i]
  while(index_min>0 and(lst_1[index_min-1]>min)):
  lst_1[index_min]=lst_1[index_min-1]
	- Dựng hàm sap_xep_chen với tham số lst là 1 list số thực.
	- Gán vị trí có giá trị nhỏ nhất là phần tử đầu tiên.
	- Nếu index_min >0 và giá trị nằm trước phần tử nhỏ nhất lớn hơn min thì gán lại phần tử có giá trị nhỏ nhất bằng phần tử trước đó, sau khi thoát khỏi

index_min-=1 lst_1[index_min]=min return lst_1	vòng lặp giá trị sẽ được cập nhật lại. Nếu không qua vòng lặp thì sẽ vẫn giữ nguyên.

Bảng 10: Mã nguồn sắp xếp chèn

5.3 Sắp xếp nổi bọt

5.3.1 Giới thiệu về sắp xếp nổi bọt

Sắp xếp nổi bọt là một giải thuật sắp xếp đơn giản. Giải thuật sắp xếp này được tiến hành dựa trên việc so sánh cặp phần tử liền kề nhau và tráo đổi thứ tự nếu chúng không theo thứ tự.

Giải thuật này không thích hợp sử dụng với các tập dữ liệu lớn khi mà độ phức tạp trường hợp xấu nhất và trường hợp trung bình là $O(n^2)$ với n là số phần tử.

5.3.2 Cài đặt sắp xếp nổi bọt

Dưới đây là mã nguồn sắp xếp nổi bọt (tăng dần).

def sap_xep_noi(lst):
lst_1=lst.copy()
for i in range(len(lst)-1):
for j in range(i+1,len(lst)):
if lst_1[i]>lst_1[j]:
lst_1[i],lst_1[j]=lst_1[j],lst_1[i]
return lst1
	- Dựng hàm sap_xep_noi với tham số lst là 1 list số thực.
	- Vòng lặp ngoài lặp từ phần tử đầu tiên đến phần từ kế cuối của danh sách.
	- Vòng lặp trong lặp từ vị trí của phần tử vòng lặp phía ngoài đến cuối danh sách.
	- Mỗi vòng lặp như vậy sẽ so sánh từng đôi một nếu phần tử phía trước lớn hơn phần tử phía sau thì hoán đổi vị trí cho nhau.

Bảng 11: Mã nguồn sắp xếp nổi bọt