# Bài 8: Thực hành cây tim kiếm nhị phân

**Giải Chuyên đề Tin học 12 Bài 8: Thực hành cây tim kiếm nhị phân**  
**Khởi động trang 37 Chuyên đề Tin học 12**: Trong Bài 7, cây tìm tìm kiếm nhị phân được cài đặt bằng mảng một chiều và mỗi nút của cây có khoá là một thuộc tính. Trong thực tế, một đối tượng có thể có nhiều thuộc tính. Ví dụ, với bài toán quản lí các món trong thực đơn, mỗi món có hai thuộc tính là tên và giá tiền. Trong trường hợp này, cây tìm kiếm nhị phân biểu diễn danh sách các món được cài đặt bằng mảng như thế nào và làm thế nào để mỗi nút của cây chứa hai thuộc tính là tên và giá tiền?  
**Lời giải:**  
Trong trường hợp mỗi nút của cây tìm kiếm nhị phân cần chứa nhiều hơn một thuộc tính, có thể sử dụng một lớp đại diện cho nút của cây, và mỗi đối tượng của lớp này sẽ chứa các thuộc tính tương ứng với mỗi nút.  
Dưới đây là một cách để triển khai sử dụng lớp để đại diện cho nút của cây tìm kiếm nhị phân trong trường hợp mỗi nút chứa hai thuộc tính là tên và giá tiền của món ăn:  
class MenuItem:  
 def \_\_init\_\_(self, name, price):  
 self.name = name  
 self.price = price  
 self.left = None  
 self.right = None  
def insert(root, name, price):  
 if root is None:  
 return MenuItem(name, price)  
 if name < root.name:  
 root.left = insert(root.left, name, price)  
 elif name > root.name:  
 root.right = insert(root.right, name, price)  
 return root  
def search(root, name):  
 if root is None or root.name == name:  
 return root  
 if name < root.name:  
 return search(root.left, name)  
 return search(root.right, name)  
# Ví dụ minh họa  
root = None  
root = insert(root, "Pho", 35)  
root = insert(root, "Banh Mi", 25)  
root = insert(root, "Bun Cha", 40)  
# Tìm kiếm một món trong cây  
search\_result = search(root, "Pho")  
if search\_result:  
 print(f"Tên món: {search\_result.name}, Giá: {search\_result.price}")  
else:  
 print("Không tìm thấy món.")  
search\_result = search(root, "Banh Xeo")  
if search\_result:  
 print(f"Tên món: {search\_result.name}, Giá: {search\_result.price}")  
else:  
 print("Không tìm thấy món.")  
**Chú thích trong mã này:**  
- Lớp MenuItem đại diện cho nút của cây, mỗi đối tượng của lớp này chứa hai thuộc tính là tên và giá tiền của món ăn, cũng như hai con trỏ left và right để tham chiếu đến các nút con bên trái và bên phải.  
- Hàm insert được sử dụng để chèn một món ăn mới vào cây tìm kiếm nhị phân.  
- Hàm search được sử dụng để tìm kiếm một món ăn trong cây tìm kiếm nhị phân.  
**Luyện tập 1 trang 40 Chuyên đề Tin học 12**: Vẽ cây tìm kiếm nhị phân ứng với tệp menu.inp trong nhiệm vụ thực hành, lưu ý mỗi nút gồm hai thuộc tính name và price.  
**Lời giải:**  
Để vẽ cây tìm kiếm nhị phân ứng với dữ liệu từ tệp **menu.inp**, trước tiên chúng ta cần đọc dữ liệu từ tệp và chèn mỗi mục vào cây. Sau đó, chúng ta có thể sử dụng các công cụ vẽ đồ thị để hiển thị cây.  
Dưới đây là mã Python mẫu để thực hiện điều này:  
import matplotlib.pyplot as plt  
from queue import Queue  
class MenuItem:  
 def \_\_init\_\_(self, name, price):  
 self.name = name  
 self.price = price  
 self.left = None  
 self.right = None  
def insert(root, name, price):  
 if root is None:  
 return MenuItem(name, price)  
 if name < root.name:  
 root.left = insert(root.left, name, price)  
 elif name > root.name:  
 root.right = insert(root.right, name, price)  
 return root  
def build\_binary\_search\_tree(filename):  
 root = None  
 with open(filename, 'r') as file:  
 for line in file:  
 name, price = line.strip().split(', ')  
 price = int(price)  
 root = insert(root, name, price)  
 return root  
def plot\_binary\_search\_tree(root):  
 if root is None:  
 return  
 node\_queue = Queue()  
 node\_queue.put(root)  
 while not node\_queue.empty():  
 current\_level\_size = node\_queue.qsize()  
 for \_ in range(current\_level\_size):  
 node = node\_queue.get()  
 if node.left:  
 node\_queue.put(node.left)  
 plt.plot([node.val, node.left.val], [node.price, node.left.price], color='black')  
 if node.right:  
 node\_queue.put(node.right)  
 plt.plot([node.val, node.right.val], [node.price, node.right.price], color='black')  
 plt.scatter(node.val, node.price, color='red')  
 plt.text(node.val, node.price, node.name, fontsize=9, ha='center')  
 plt.xlabel('Name')  
 plt.ylabel('Price')  
 plt.title('Binary Search Tree Representation of Menu')  
 plt.grid()  
 plt.show()  
# Tạo cây tìm kiếm nhị phân từ tệp menu.inp  
root = build\_binary\_search\_tree('menu.inp')  
# Vẽ cây tìm kiếm nhị phân  
plot\_binary\_search\_tree(root)  
Chú thích trong chương trình này:  
- Chúng ta định nghĩa lớp MenuItem để đại diện cho các nút trong cây. Mỗi nút có hai thuộc tính là name và price.  
- Hàm insert được sử dụng để chèn một mục vào cây tìm kiếm nhị phân.  
- Hàm build\_binary\_search\_tree đọc dữ liệu từ tệp menu.inp và xây dựng cây tìm kiếm nhị phân từ các mục đó.  
- Hàm plot\_binary\_search\_tree được sử dụng để vẽ cây tìm kiếm nhị phân bằng cách sử dụng thư viện matplotlib.  
**Luyện tập 2 trang 40 Chuyên đề Tin học 12**: Mô tả quá trình tra cứu giá tiền món Bún chả thực hiện trên cây tìm kiếm nhị phân đã vẽ ở Luyện tập 1.  
**Lời giải:**  
Quá trình tra cứu giá tiền của món "Bún chả" trên cây tìm kiếm nhị phân đã vẽ có thể được mô tả như sau:  
1. Bắt đầu từ nút gốc của cây.  
2. So sánh tên của nút gốc với "Bún chả":  
- Nếu tên của nút gốc là "Bún chả", chúng ta đã tìm thấy món "Bún chả". Lúc này, chúng ta lấy giá tiền của món "Bún chả" từ nút này.  
- Nếu tên của nút gốc lớn hơn "Bún chả", chúng ta di chuyển sang cây con bên trái của nút gốc và tiếp tục tìm kiếm.  
- Nếu tên của nút gốc nhỏ hơn "Bún chả", chúng ta di chuyển sang cây con bên phải của nút gốc và tiếp tục tìm kiếm.  
3. Quá trình này lặp lại cho đến khi chúng ta tìm thấy món "Bún chả" hoặc điều kiện dừng được kích hoạt (không có nút nào chứa tên "Bún chả").  
Nếu món "Bún chả" được tìm thấy, chúng ta sẽ biết được giá tiền của món này. Nếu không, chúng ta sẽ biết rằng món "Bún chả" không có trong danh sách thực đơn và không có thông tin về giá tiền.  
**Vận dụng 1 trang 40 Chuyên đề Tin học 12**: Sử dụng cây tìm kiếm nhị phân để viết ứng dụng quản lí tài khoản ngân hàng. Mỗi một tài khoản gồm mã tài khoản (duy nhất) và số dư tài khoản. Ứng dụng cho phép thêm tài khoản, sửa số dư tài khoản, tìm kiếm tài khoản theo mã tài khoản.  
**Lời giải:**  
Hướng dẫn gợi ý về cách sử dụng cây tìm kiếm nhị phân để viết ứng dụng quản lí tài khoản ngân hàng thông qua một chương trình mẫu sau:   
class Account:  
 def \_\_init\_\_(self, account\_id, balance):  
 self.account\_id = account\_id  
 self.balance = balance  
 self.left = None  
 self.right = None  
class BankAccountManager:  
 def \_\_init\_\_(self):  
 self.root = None  
 def insert(self, root, account\_id, balance):  
 if root is None:  
 return Account(account\_id, balance)  
 if account\_id < root.account\_id:  
 root.left = self.insert(root.left, account\_id, balance)  
 elif account\_id > root.account\_id:  
 root.right = self.insert(root.right, account\_id, balance)  
 return root  
 def add\_account(self, account\_id, balance):  
 self.root = self.insert(self.root, account\_id, balance)  
 def search(self, root, account\_id):  
 if root is None or root.account\_id == account\_id:  
 return root  
 if account\_id < root.account\_id:  
 return self.search(root.left, account\_id)  
 return self.search(root.right, account\_id)  
 def find\_account(self, account\_id):  
 return self.search(self.root, account\_id)  
 def update\_balance(self, account\_id, new\_balance):  
 account = self.find\_account(account\_id)  
 if account:  
 account.balance = new\_balance  
 else:  
 print("Account not found.")  
# Sử dụng ứng dụng  
bank\_manager = BankAccountManager()  
# Thêm tài khoản  
bank\_manager.add\_account(1001, 50000)  
bank\_manager.add\_account(1002, 100000)  
bank\_manager.add\_account(1003, 75000)  
# Tìm kiếm tài khoản  
account = bank\_manager.find\_account(1002)  
if account:  
 print(f"Account found - Account ID: {account.account\_id}, Balance: {account.balance}")  
else:  
 print("Account not found.")  
# Cập nhật số dư tài khoản  
bank\_manager.update\_balance(1001, 60000)  
# Kiểm tra lại số dư sau khi cập nhật  
account = bank\_manager.find\_account(1001)  
if account:  
 print(f"Updated balance - Account ID: {account.account\_id}, Balance: {account.balance}")  
else:  
 print("Account not found.")  
Chú thích trong chương trình này:  
- Lớp Account được sử dụng để đại diện cho mỗi tài khoản, với hai thuộc tính là account\_id (mã tài khoản) và balance (số dư tài khoản).  
- Lớp BankAccountManager chứa các phương thức để thêm tài khoản, tìm kiếm tài khoản và cập nhật số dư tài khoản. Mỗi tài khoản được lưu trữ trong một cây tìm kiếm nhị phân, với account\_id làm khóa.  
**Vận dụng 2 trang 40 Chuyên đề Tin học 12**: Sử dụng cây tìm kiếm nhị phân để viết chương trình quản lí danh sách học sinh của một trường trung học phổ thông. Mỗi một học sinh gồm các thông tin: mã học sinh (duy nhất), họ tên, ngày sinh. Chương trình cho phép thêm một học sinh vào danh sách với các trường thông tin kể trên, tìm kiếm học sinh theo mã và sửa đổi họ tên và ngày sinh ứng với một mã học sinh.  
**Lời giải:**  
Gợi ý chương trình mẫu về cách sử dụng cây tìm kiếm nhị phân để quản lí danh sách học sinh của một trường trung học phổ thông:  
class Student:  
 def \_\_init\_\_(self, student\_id, name, date\_of\_birth):  
 self.student\_id = student\_id  
 self.name = name  
 self.date\_of\_birth = date\_of\_birth  
 self.left = None  
 self.right = None  
class StudentManager:  
 def \_\_init\_\_(self):  
 self.root = None  
 def insert(self, root, student\_id, name, date\_of\_birth):  
 if root is None:  
 return Student(student\_id, name, date\_of\_birth)  
 if student\_id < root.student\_id:  
 root.left = self.insert(root.left, student\_id, name, date\_of\_birth)  
 elif student\_id > root.student\_id:  
 root.right = self.insert(root.right, student\_id, name, date\_of\_birth)  
 return root  
 def add\_student(self, student\_id, name, date\_of\_birth):  
 self.root = self.insert(self.root, student\_id, name, date\_of\_birth)  
 def search(self, root, student\_id):  
 if root is None or root.student\_id == student\_id:  
 return root  
 if student\_id < root.student\_id:  
 return self.search(root.left, student\_id)  
 return self.search(root.right, student\_id)  
 def find\_student(self, student\_id):  
 return self.search(self.root, student\_id)  
 def update\_student(self, student\_id, new\_name, new\_date\_of\_birth):  
 student = self.find\_student(student\_id)  
 if student:  
 student.name = new\_name  
 student.date\_of\_birth = new\_date\_of\_birth  
 else:  
 print("Student not found.")  
# Sử dụng chương trình  
student\_manager = StudentManager()  
# Thêm học sinh  
student\_manager.add\_student(1001, "Nguyen Van A", "2005-03-15")  
student\_manager.add\_student(1002, "Tran Thi B", "2004-08-22")  
student\_manager.add\_student(1003, "Le Van C", "2006-01-10")  
# Tìm kiếm học sinh  
student = student\_manager.find\_student(1002)  
if student:  
 print(f"Student found - Student ID: {student.student\_id}, Name: {student.name}, Date of Birth: {student.date\_of\_birth}")  
else:  
 print("Student not found.")  
# Cập nhật thông tin học sinh  
student\_manager.update\_student(1001, "Pham Thi D", "2005-05-20")  
# Kiểm tra lại thông tin sau khi cập nhật  
student = student\_manager.find\_student(1001)  
if student:  
 print(f"Updated student - Student ID: {student.student\_id}, Name: {student.name}, Date of Birth: {student.date\_of\_birth}")  
else:  
 print("Student not found.")  
Lưu ý trong mã chương trình này:  
- Lớp Student đại diện cho mỗi học sinh, với các thuộc tính là student\_id (mã học sinh), name (họ tên) và date\_of\_birth (ngày sinh).  
- Lớp StudentManager chứa các phương thức để thêm học sinh vào danh sách, tìm kiếm học sinh theo mã và cập nhật thông tin học sinh. Mỗi học sinh được lưu trữ trong một cây tìm kiếm nhị phân, với student\_id làm khóa.