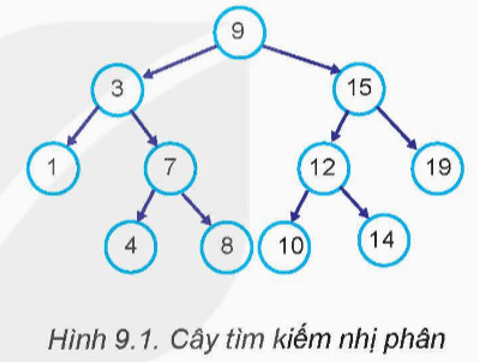
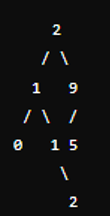
# Bài 9: Các thuật toán duyệt trên cây tìm kiếm nhị phân

**Giải Chuyên đề Tin học 12 Bài 9: Các thuật toán duyệt trên cây tìm kiếm nhị phân**  
**Khởi động trang 41 Chuyên đề Tin học 12**: Quan sát cây tìm kiếm nhị phân trong hình 9.1, cùng trao đổi, thảo luận các câu hỏi sau:  
a) Nếu thực hiện thuật toán duyệt giữa (trái – gốc – phải) thì nút đầu tiên được duyệt là nút nào?  
b) Nút cuối cùng được duyệt là nút nào?  
c) Thứ tự các nút được duyệt theo thuật toán duyệt giữa sẽ theo thứ tự nào? Em có nhận xét gì về kết quả đạt được? Giải thích vì sao.  
  
**Lời giải:**  
a) Nút đầu tiên: Khi thực hiện thuật toán duyệt giữa, nút đầu tiên được duyệt là nút ‘1’, vì nó là nút nằm xa nhất về bên trái của cây.  
b) Nút cuối cùng: Nút cuối cùng được duyệt là nút ‘19’, vì nó là nút nằm xa nhất về bên phải của cây.  
c) Thứ tự duyệt: Các nút sẽ được duyệt theo thứ tự sau: 1, 3, 4, 7, 8, 9, 10, 12, 15, 14, 19. Kết quả này cho thấy thuật toán duyệt giữa trên cây tìm kiếm nhị phân sẽ cho ra danh sách các nút theo thứ tự tăng dần. Điều này xảy ra bởi vì thuật toán này luôn tuân theo quy tắc: trước tiên duyệt tất cả các nút nhỏ hơn nút gốc (bên trái), sau đó là nút gốc (ở giữa), và cuối cùng là tất cả các nút lớn hơn nút gốc (bên phải).  
**Hoạt động 1 trang 41 Chuyên đề Tin học 12**: Quan sát cách cài đặt các thuật toán duyệt trên cây tìm kiếm nhị phân và trao đổi về ý nghĩa và sự khác biệt khi thực hiện các thuật toán này so với các thuật toán duyệt cây nhị phân đã học trong Bài 6.  
**Lời giải:**  
Ý nghĩa của các thuật toán duyệt trên cây tìm kiếm nhị phân:  
- Duyệt trước: Dùng để sao chép cây, tính toán giá trị các nút trong trường hợp biểu thức toán học cây biểu thức, hay xử lý cây trong nhiều ứng dụng khác.  
- Duyệt giữa: Đặc biệt hữu ích cho BST vì nó trả về các giá trị theo thứ tự tăng dần. Đây là lý do tại sao in-order traversal được sử dụng phổ biến trong các ứng dụng yêu cầu dữ liệu có thứ tự.  
- Duyệt sau: Thường dùng trong việc xóa cây hoặc giải phóng bộ nhớ vì nó đảm bảo rằng một nút chỉ được thăm sau khi các cây con của nó đã được xử lý.  
Sự khác biệt giữa các thuật toán duyệt trên cây tìm kiếm nhị phân (trong BST) so với cây nhị phân đã học ở bài 6 ( cây nhị phân hoàn chỉnh):  
- Khi duyệt BST biểu diễn bằng mảng, cần kiểm tra thêm điều kiện k < len(T) và T[k] is not None để tránh xử lý các nút giả None, đảm bảo không thăm các nút không tồn tại.  
- Các thuật toán duyệt cơ bản không thay đổi về bản chất, nhưng cần chú ý đến việc xử lý các nút giả để tránh lỗi ngoài ý muốn.  
**Câu hỏi 1 trang 43 Chuyên đề Tin học 12**: Cho trước dãy số A = [2,1,9,0,2,1,5]. Tạo cây tìm kiếm nhị phân T từ dãy A và thực hiện thuật toán duyệt giữa trên cây T. Em hãy cho biết kết quả duyệt là dãy các khoá có thứ tự như thế nào.  
**Lời giải:**  
Các bước xây dựng cây tìm kiếm nhị phân (BST) từ dãy số A  
**- Chèn 2**: Làm gốc của cây.  
**- Chèn 1**: So với 2, 1 nhỏ hơn, chèn vào bên trái của 2.  
**- Chèn 9**: So với 2, 9 lớn hơn, chèn vào bên phải của 2.  
**- Chèn 0**: So với 2, nhỏ hơn; so với 1, nhỏ hơn, chèn vào bên trái của 1.  
**- Chèn 2 (thứ hai)**: So với 2, bằng nhau; theo quy tắc chung (trường hợp đặc biệt), chèn vào bên phải của 2 gốc  
**- Chèn 1 (thứ hai)**: So với 2, nhỏ hơn; so với 1, bằng nhau; chèn vào bên phải của 1.  
**- Chèn 5**: So với 2, lớn hơn; so với 9, nhỏ hơn, chèn vào bên trái của 9.  
**Cấu trúc cây BST**  
  
Duyệt giữa (In-order Traversal)  
Duyệt giữa trên cây tìm kiếm nhị phân là duyệt theo thứ tự cây con trái, nút gốc, cây con phải.  
Cụ thể các bước thực hiện như sau:  
- Duyệt cây con trái của nút gốc (2):  
+ Duyệt cây con trái của nút 1:  
+ Duyệt cây con trái của 0 (không có nút con trái nào), thăm 0.  
+ Thăm nút 1.  
+ Duyệt cây con phải của 1:  
2. Duyệt cây con trái của 1 (thứ hai) (không có nút con trái nào), thăm 1 (thứ hai).  
- Thăm nút 1.  
- Thăm nút gốc (2).  
- Duyệt cây con phải của nút gốc (2):  
- Duyệt cây con trái của nút 9:  
- Duyệt cây con trái của 5 (không có nút con trái nào), thăm 5.  
- Thăm nút 9.  
- Duyệt cây con phải của 9 (không có nút con phải nào).  
Kết quả duyệt giữa  
0, 1, 1, 2, 2, 5, 9  
Như vậy, dãy các khoá theo thứ tự duyệt giữa trên cây tìm kiếm nhị phân được tạo từ dãy A = [2,1,9,0,2,1,5] là: [0, 1, 1, 2, 2, 5, 9].  
**Câu hỏi 2 trang 43 Chuyên đề Tin học 12**: Với cây T như Câu 1, nếu thực hiện thuật toán duyệt ngược thì thứ tự các khoá thể hiện trên màn hình như thế nào?  
**Lời giải:**  
Hướng dẫn cụ thể các bước thực hiện như sau:  
1. Duyệt cây con phải của nút gốc (2):  
- Duyệt cây con phải của nút 9 (không có nút con phải nào).  
- Thăm nút 9.  
- Duyệt cây con trái của nút 9:  
Duyệt cây con phải của nút 5 (không có nút con phải nào).  
Thăm nút 5.  
Duyệt cây con trái của nút 5 (không có nút con trái nào).  
2. Thăm nút gốc (2).  
3. Duyệt cây con trái của nút gốc (2):  
Duyệt cây con phải của nút 1:  
- Duyệt cây con phải của nút 1 (thứ hai) (không có nút con phải nào).  
- Thăm nút 1 (thứ hai).  
- Duyệt cây con trái của nút 1 (thứ hai) (không có nút con trái nào).  
Thăm nút 1.  
Duyệt cây con trái của nút 1:  
- Duyệt cây con phải của nút 0 (không có nút con phải nào).  
- Thăm nút 0.  
- Duyệt cây con trái của nút 0 (không có nút con trái nào).  
**Kết quả duyệt ngược**  
9, 5, 2, 2, 1, 1, 0  
Như vậy, dãy các khoá theo thứ tự duyệt ngược trên cây tìm kiếm nhị phân được tạo từ dãy A = [2,1,9,0,2,1,5] là: **[9, 5, 2, 2, 1, 1, 0]**.  
**2. Sắp xếp dãy số bằng cây tìm kiếm nhị phân**  
**Hoạt động 2 trang 43 Chuyên đề Tin học 12**: Trao đổi, thảo luận để giải bài toán sau:  
Cho trước dãy số A. Thiết kế thuật toán sắp xếp lại dãy A theo thứ tự tăng dần hoặc giảm dần.  
**Lời giải:**  
Để sắp xếp dãy số A theo thứ tự tăng dần hoặc giảm dần, chúng ta có thể sử dụng nhiều thuật toán sắp xếp khác nhau. Một trong những phương pháp hiệu quả và dễ hiểu là sử dụng cây tìm kiếm nhị phân (BST) để sắp xếp dãy số.  
Để sắp xếp dãy số A theo thứ tự tăng dần hoặc giảm dần, chúng ta có thể sử dụng nhiều thuật toán sắp xếp khác nhau. Một trong những phương pháp hiệu quả và dễ hiểu là sử dụng cây tìm kiếm nhị phân (BST) để sắp xếp dãy số.  
Ý tưởng thuật toán  
1. Chèn các phần tử của dãy A vào cây tìm kiếm nhị phân (BST).  
2. Duyệt cây theo thứ tự tăng dần hoặc giảm dần để lấy ra các phần tử đã được sắp xếp.  
Thuật toán chi tiết:  
1. Chèn các phần tử vào BST  
Mỗi phần tử trong dãy A sẽ được chèn vào BST theo quy tắc:  
- Nếu phần tử nhỏ hơn nút hiện tại, chuyển sang cây con trái.  
- Nếu phần tử lớn hơn hoặc bằng nút hiện tại, chuyển sang cây con phải.  
2. Duyệt cây để lấy ra dãy đã sắp xếp  
- Duyệt giữa (In-order Traversal): để lấy các phần tử theo thứ tự tăng dần.  
- Duyệt ngược (Reverse in-order Traversal): để lấy các phần tử theo thứ tự giảm dần.  
Nhận xét:  
- Hiệu quả: Thuật toán sử dụng BST có thời gian chèn và tìm kiếm trung bình làkhi cây cân bằng, và O(n)O(n)O(n) trong trường hợp xấu nhất khi cây trở thành đường thẳng (skewed tree).  
- Tính chất: BST duy trì các phần tử theo thứ tự, do đó việc duyệt cây sẽ cho ra dãy số đã được sắp xếp.  
- Ứng dụng: Thuật toán này không phù hợp cho các trường hợp cần sắp xếp nhanh và hiệu quả như QuickSort hoặc MergeSort, nhưng nó giúp minh họa tốt về cách sử dụng cấu trúc cây trong việc sắp xếp dữ liệu.  
**Câu hỏi 1 trang 45 Chuyên đề Tin học 12**: Viết lại hàm BSTSort(A) thực hiện sắp xếp dãy số A theo thứ tự tăng dần nhưng kết quả không cập nhật vào A. Hàm trả lại dãy số mới là dãy vừa được sắp xếp (gồm các phần tử của dãy A).  
**Lời giải:**  
**1. Định nghĩa cấu trúc nút cây BST:**  
class TreeNode:  
 def \_\_init\_\_(self, key):  
 self.left = None  
 self.right = None  
 self.val = key  
**2. Hàm chèn một phần tử vào BST**  
def insert(root, key):  
 if root is None:  
 return TreeNode(key)  
 else:  
 if root.val < key:  
 root.right = insert(root.right, key)  
 else:  
 root.left = insert(root.left, key)  
 return root  
**3. Hàm duyệt giữa để lấy thứ tự tăng dần**  
def in\_order\_traversal(root, result):  
 if root:  
 in\_order\_traversal(root.left, result)  
 result.append(root.val)  
 in\_order\_traversal(root.right, result)  
**4. Hàm chính để sắp xếp dãy A**  
def BSTSort(A):  
 if not A:  
 return []  
 # Bước 1: Tạo cây tìm kiếm nhị phân từ dãy A  
 root = None  
 for key in A:  
 root = insert(root, key)  
 # Bước 2: Duyệt cây để lấy kết quả sắp xếp  
 sorted\_result = []  
 in\_order\_traversal(root, sorted\_result)  
 return sorted\_result  
**Câu hỏi 2 trang 45 Chuyên đề Tin học 12**: Nếu không cần cập nhật vào dãy A mà chỉ cần in ra màn hình các phần tử của A theo thứ tự tăng dần thì cần sửa lại chương trình sắp xếp trên như thế nào?  
**Lời giải:**  
Nếu không cần cập nhật vào dãy A mà chỉ cần in ra màn hình các phần tử của A theo thứ tự tăng dần, bạn có thể sửa lại chương trình sắp xếp trên để thay vì trả về dãy số mới, nó sẽ in trực tiếp các phần tử theo thứ tự tăng dần trong quá trình duyệt cây.  
Cài đặt lại hàm để in trực tiếp  
1. Định nghĩa cấu trúc nút cây BST  
class TreeNode:  
 def \_\_init\_\_(self, key):  
 self.left = None  
 self.right = None  
 self.val = key  
2. Hàm chèn một phần tử vào BST  
def insert(root, key):  
 if root is None:  
 return TreeNode(key)  
 else:  
 if root.val < key:  
 root.right = insert(root.right, key)  
 else:  
 root.left = insert(root.left, key)  
 return root  
3. Hàm duyệt giữa để in các phần tử theo thứ tự tăng dần  
def in\_order\_traversal\_and\_print(root):  
 if root:  
 in\_order\_traversal\_and\_print(root.left)  
 print(root.val, end=' ')  
 in\_order\_traversal\_and\_print(root.right)  
4. Hàm chính để sắp xếp và in dãy A  
def BSTSortAndPrint(A):  
 if not A:  
 return  
 # Bước 1: Tạo cây tìm kiếm nhị phân từ dãy A  
 root = None  
 for key in A:  
 root = insert(root, key)  
 # Bước 2: Duyệt cây để in các phần tử theo thứ tự tăng dần  
 in\_order\_traversal\_and\_print(root)  
 print() # Thêm dòng mới sau khi in xong  
**Luyện tập 1 trang 45 Chuyên đề Tin học 12**: Dựa trên hàm BSTSort(A) đã biết, viết chương trình sắp xếp dãy số giảm dần theo kĩ thuật sử dụng cây tìm kiếm nhị phân.  
**Lời giải:**  
Để sắp xếp dãy số theo thứ tự giảm dần bằng cách sử dụng cây tìm kiếm nhị phân (BST), chúng ta cần thực hiện duyệt ngược (reverse in-order traversal) trên cây. Điều này có nghĩa là chúng ta sẽ duyệt cây theo thứ tự: cây con phải, nút gốc, cây con trái.  
Cài đặt hàm BSTSort để sắp xếp giảm dần  
**1. Định nghĩa cấu trúc nút cây BST**  
class TreeNode:  
 def \_\_init\_\_(self, key):  
 self.left = None  
 self.right = None  
 self.val = key  
**2. Hàm chèn một phần tử vào BST**  
def insert(root, key):  
 if root is None:  
 return TreeNode(key)  
 else:  
 if root.val < key:  
 root.right = insert(root.right, key)  
 else:  
 root.left = insert(root.left, key)  
 return root  
**3. Hàm duyệt ngược để lấy thứ tự giảm dần**  
def reverse\_in\_order\_traversal(root, result):  
 if root:  
 reverse\_in\_order\_traversal(root.right, result)  
 result.append(root.val)  
 reverse\_in\_order\_traversal(root.left, result)  
**4. Hàm chính để sắp xếp dãy A theo thứ tự giảm dần**  
def BSTSortDescending(A):  
 if not A:  
 return []  
 # Bước 1: Tạo cây tìm kiếm nhị phân từ dãy A  
 root = None  
 for key in A:  
 root = insert(root, key)  
 # Bước 2: Duyệt cây để lấy kết quả sắp xếp giảm dần  
 sorted\_result = []  
 return sorted\_result  
**Luyện tập 2 trang 45 Chuyên đề Tin học 12**: Thuật toán sắp xếp dãy sử dụng cây tìm kiếm nhị phân có độ phức tạp thời gian là bao nhiêu?  
**Lời giải:**  
Thuật toán sắp xếp dãy sử dụng cây tìm kiếm nhị phân (BST) có độ phức tạp thời gian phụ thuộc vào cấu trúc của cây BST trong quá trình chèn và duyệt các phần tử.  
Độ phức tạp thời gian của thuật toán sắp xếp bằng BST  
1. Chèn phần tử vào BST:  
- Trung bình: Đối với cây BST cân bằng, độ sâu trung bình của cây là  do đó, việc chèn mỗi phần tử có độ phức tạp trung bình là   
- Trường hợp xấu nhất: Trong trường hợp xấu nhất, nếu cây BST trở thành một cây một nhánh (giống như danh sách liên kết) khi các phần tử được chèn theo thứ tự tăng hoặc giảm dần, độ sâu của cây sẽ là O(n)O(n)O(n). Vì vậy, việc chèn mỗi phần tử có độ phức tạp là O(n)O(n)O(n).  
2. Duyệt cây để lấy các phần tử đã sắp xếp:  
- Việc duyệt cây (in-order hoặc reverse in-order) có độ phức tạp là O(n)O(n)O(n) vì chúng ta phải thăm tất cả các nút trong cây một lần.  
Tổng độ phức tạp thời gian  
- Trung bình: Trong trường hợp trung bình khi cây BST gần như cân bằng, độ phức tạp cho việc chèn n phần tử là và duyệt cây là O(n)O(n)O(n). Do đó, tổng độ phức tạp thời gian là   
- Trường hợp xấu nhất: Trong trường hợp xấu nhất khi cây BST trở thành một cây một nhánh, độ phức tạp cho việc chèn n phần tử là O(n2)O(n^2)O(n2), và duyệt cây là O(n)O(n)O(n). Do đó, tổng độ phức tạp thời gian là O(n2)O(n^2)O(n2).  
Kết luận  
- Trung bình:   
- Trường hợp xấu nhất: O(n2)O(n^2)O(n2)  
Để tránh trường hợp xấu nhất, các thuật toán như AVL tree hoặc Red-Black tree có thể được sử dụng để đảm bảo rằng cây BST luôn gần như cân bằng, giữ độ phức tạp thời gian ở mức   
**Vận dụng 1 trang 45 Chuyên đề Tin học 12**: Dựa trên tính chất của cây tìm kiếm nhị phân, hãy viết hàm minimum(T) và maximum(T) tính giá trị khoá nhỏ nhất và lớn nhất của cây tìm kiếm nhị phân T.  
**Lời giải:**  
Dựa trên tính chất của cây tìm kiếm nhị phân (BST), giá trị nhỏ nhất của cây sẽ nằm ở nút lá bên trái cùng (nếu có) và giá trị lớn nhất sẽ nằm ở nút lá bên phải cùng (nếu có).  
Dưới đây là cài đặt Python cho hai hàmminimum(T) và maximum(T) để tính giá trị khoá nhỏ nhất và lớn nhất của cây tìm kiếm nhị phân T.  
class TreeNode:  
 def \_\_init\_\_(self, key):  
 self.left = None  
 self.right = None  
 self.val = key  
def minimum(T):  
 # Duyệt qua các nút con bên trái cho đến khi không còn nút con nào nữa  
 while T.left is not None:  
 T = T.left  
 return T.val  
def maximum(T):  
 # Duyệt qua các nút con bên phải cho đến khi không còn nút con nào nữa  
 while T.right is not None:  
 T = T.right  
 return T.val  
Giải thích:  
- Hàm minimum(T) sẽ duyệt qua cây từ nút gốc theo hướng sang trái cho đến khi không còn nút con nào bên trái nữa. Khi đó, nút hiện tại sẽ là nút có giá trị khoá nhỏ nhất của cây.  
- Tương tự, hàm maximum(T) sẽ duyệt qua cây từ nút gốc theo hướng sang phải cho đến khi không còn nút con nào bên phải nữa. Khi đó, nút hiện tại sẽ là nút có giá trị khoá lớn nhất của cây.  
**Vận dụng 2 trang 45 Chuyên đề Tin học 12**: Viết hàm height(T) tính chiều cao của cây tìm kiếm nhị phân T.  
**Lời giải:**  
Để tính chiều cao của cây tìm kiếm nhị phân (BST), chúng ta có thể sử dụng phương pháp đệ quy. Chiều cao của một cây BST là độ dài của đường dẫn từ nút gốc đến nút lá xa nhất. Dưới đây là cài đặt Python cho hàm height(T) để tính chiều cao của cây tìm kiếm nhị phân T.  
class TreeNode:  
 def \_\_init\_\_(self, key):  
 self.left = None  
 self.right = None  
 self.val = key  
def height(T):  
 if T is None:  
 return -1 # Chiều cao của một cây rỗng là -1  
 else:  
 # Tính chiều cao của cây con bên trái và cây con bên phải  
 left\_height = height(T.left)  
 right\_height = height(T.right)  
 # Chiều cao của cây là chiều cao lớn nhất của hai cây con cộng thêm 1  
 return max(left\_height, right\_height) + 1  
**Giải thích:**  
- Hàm height(T) sẽ tính chiều cao của cây tìm kiếm nhị phân T bằng cách sử dụng đệ quy.  
- Nếu cây T là cây rỗng (None), chiều cao của cây là -1.  
- Nếu cây T không rỗng, chúng ta tính chiều cao của cây con bên trái và cây con bên phải.  
- Chiều cao của cây là chiều cao lớn nhất của hai cây con cộng thêm 1 (chiều cao của nút gốc).