**BỘ CÔNG THƯƠNG**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THỰC PHẨM TP.HCM**

**KHOA CÔNG NGHỆ THÔNG TIN**

**---🙞🕗🙜---**

Ảnh có chứa biểu tượng, Nhãn hiệu, văn bản, Phông chữ

Mô tả được tạo tự động

**ĐỒ ÁN CUỐI KỲ MÔN**

**CTDL & GT**

GVHD: Trần Văn Thọ

Thực hiện : Nhóm 6

*Tp.HCM 2025*

# **NHẬT KÝ LÀM VIỆC CỦA NHÓM**

**Lớp học phần:** 15DHBM03 **Nhóm: 6 Mã đề tài: 11**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Nội dung** | **Thời gian** | **Họ và tên** | **MSSV** | **Đã thực hiện** | **Ký tên** | **Nhận xét của nhóm trưởng** |
| Phân công việc | 28/04/2025 | Nguyễn Thành Nam | **2001240286** | Chia nhỏ các phần và phân công công việc cho nhóm. |  |  |
| Thu thập thông tin, giải các bài tập  (đóng file word gửi nhóm trưởng) | 29/04/2025 | Trần Nguyễn Vỹ Kha | **2033240132** | Xây dựng cấu trúc dữ liệu HocPhan, viết hàm tạo cây bằng cách nhập từ bàn phím, đọc dữ liệu từ file và hàm insert. |  |  |
| 02/05/2025 | Nguyễn Thành Nam | **2001240286** | Cài đặt hàm tìm kiếm học phần theo tên, thêm học phần X vào cây, cài đặt hàm xóa học phần và 2 hàm duyệt cây theo chiều rộng và chiều sâu. |  |  |
| 02/05/2025 | Đào Sỹ Luận | **2001240265** | Cài đặt 6 hàm duyệt cây, hàm thống kê học phần theo tín chỉ, hàm đếm số lượng học phần theo từng loại, tính tổng số tín chỉ các học phần. |  |  |
| 05/05/2025 | Phạm Gia Hoàng Lâm | **2033240172** | Tìm hiểu BST trên Python và cài đặt BST, hàm thêm, tìm kiếm và các cách duyệt cây trên Python. |  |  |
| 07/05/2025 | Nguyễn Duy Bảo | **2033240028** | Trình bày khái niệm, diễn giải các thao tác cơ bản trên Python và so sánh hiệu suất của BST với List, AVL, Red-Black Tree. |  |  |
| Thiết kế word | 04/05/2025 | Nguyễn Thành Nam | **2001240286** | Kiểm duyệt các file và thuyết kế word. |  |  |
| 06/05/2025 | Nguyễn Thành Nam | **2001240286** | Thêm phần của Hoàng Lâm vào word. |  |  |
| 07/05/2025 | Nguyễn Thành Nam | **2001240286** | Thêm phần của Nguyễn Duy Bảo vào word. |  |  |

# **ĐÁNH GIÁ CÁ NHÂN TRONG LÀM VIỆC NHÓM**

*(Đánh giá theo tiêu chí như Rubric 4)*

**Lớp học phần:** 15DHBM03 **Nhóm: 6 Mã đề tài: 11**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Họ và tên** | **MSSV** | **Công việc được giao** | **Thời gian tham gia họp nhóm đầy đủ**  **(15**%**)** | **Thái độ tham gia tích cực**  **(15**%**)** | **Ý kiến đóng góp hữu ích**  **(20**%**)** | **Thời gian giao nộp sản phẩm đúng hạn**  **(20**%**)** | **Chất lượng sản phẩm giao nộp tốt**  **(30**%**)** | **Tổng mức độ đóng góp (%)** | **Ký tên** |
| Trần Nguyễn Vỹ Kha | **2033240132** | Xây dựng cấu trúc dữ liệu HocPhan, viết hàm tạo cây bằng cách nhập từ bàn phím, đọc dữ liệu từ file và hàm insert, làm PowerPoint. | 15% | 15% | 20% | 20% | 30% | **100%** |  |
| Nguyễn Thành Nam | **2001240286** | Cài đặt hàm tìm kiếm học phần theo tên, thêm học phần X vào cây, cài đặt hàm xóa học phần và 2 hàm duyệt cây theo chiều rộng và chiều sâu, làm Word. | 15% | 15% | 20% | 20% | 30% | **100%** |  |
| Đào Sỹ Luận | **2001240265** | Cài đặt 6 hàm duyệt cây, hàm thống kê học phần theo tín chỉ, hàm đếm số lượng học phần theo từng loại, tính tổng số tín chỉ các học phần. | 15% | 15% | 20% | 20% | 30% | **100%** |  |
| Phạm Gia Hoàng Lâm | **2033240172** | Tìm hiểu BST trên Python và cài đặt BST, hàm thêm, tìm kiếm và các cách duyệt cây trên Python. | 15% | 15% | 20% | 20% | 30% | **100%** |  |
| Nguyễn Duy Bảo | **2033240028** | Trình bày khái niệm, diễn giải các thao tác cơ bản trên Python và so sánh hiệu suất của BST với List, AVL, Red-Black Tree. | 15% | 15% | 20% | 20% | 30% | **100%** |  |

# **NHẬN XÉT CỦA GIẢNG VIÊN VỀ TIỂU LUẬN**

*(Đánh giá theo tiêu chí như Rubric 5)*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Tiêu chí**  **đánh giá** | **CĐR** | **Trọng số**  ***(%)*** | **Thang điểm** | | | |
| **Tốt**  **(100%)** | **Khá**  **(75%)** | **Trung bình**  **(50%)** | **Kém**  **(0%)** |
| 1. Cấu trúc hợp lý |  | 5 |  |  |  |  |
| 2. Nội dung đầy đủ |  | 85 |  |  |  |  |
| 3. Hình thức trình bày chuẩn mực (font chữ, canh lề, format, lỗi chính tả) |  | 10 |  |  |  |  |
| **Tổng điểm** | | **100%** |  | | | |

**MỨC ĐỘ ĐÓNG GÓP VÀ ĐIỂM CỦA CÁC THÀNH VIÊN**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TT** | **Họ và tên** | **Mức độ đóng góp**  *(Nhóm trưởng)* | **Điểm tiểu luận**  *(GV)* | **Điểm quá trình của cá nhân** *(GV)* |
|  | Nguyễn Thành Nam | 23% |  |  |
|  | Trần Nguyễn Vỹ Kha | 23% |  |  |
|  | Đào Sỹ Luận | 18% |  |  |
|  | Phạm Gia Hoàng Lâm | 18% |  |  |
|  | Nguyễn Duy Bảo | 18% |  |  |

*Thành phố Hồ Chí Minh, ngày…… tháng …… năm 2025*

**Giảng viên hướng dẫn**

# **LỜI CẢM ƠN**

Em xin chân thành cảm ơn thầy, cô khoa Công nghệ thông tin trường Đại học Công thương, đặc biệt là thầy Trần Văn Thọ đã giảng dạy tận tâm và cung cấp những kiến thức nền tảng vững chắc về Cấu trúc dữ liệu và Giải thuật. Sự hướng dẫn nhiệt tình và những góp ý sâu sắc của thầy đã giúp em hiểu rõ hơn về môn học và hoàn thành bài báo cáo này.

Em cũng cảm ơn bạn bè đã hỗ trợ, chia sẻ tài liệu và kinh nghiệm để hoàn thành tốt quá trình học tập. Những kiến thức từ môn học này chắc chắn quan trọng cho con đường học tập trong tương lai.

Cuối cùng, em xin gửi lời tri ân sâu sắc và chúc thầy thật nhiều sức khỏe, thành công trong sự nghiệp giảng dạy và nghiên cứu.

# **Mục lục**

[**NHẬT KÝ LÀM VIỆC CỦA NHÓM** i](#_Toc200310638)

[**ĐÁNH GIÁ CÁ NHÂN TRONG LÀM VIỆC NHÓM** iii](#_Toc200310639)

[**NHẬN XÉT CỦA GIẢNG VIÊN VỀ TIỂU LUẬN** v](#_Toc200310640)

[**LỜI CẢM ƠN** 1](#_Toc200310641)

[**Mục lục** 2](#_Toc200310642)

[**CHƯƠNG I: TỔNG QUAN** 4](#_Toc200310643)

[**1.** **Binary Search Tree trong Python:** 4](#_Toc200310644)

[1.1. Khái niệm: 4](#_Toc200310645)

[1.2. Các thao tác cơ bản của Binary Search Tree: 4](#_Toc200310646)

[a) Chèn các phần tử vào cây: 4](#_Toc200310647)

[b) Tìm kiếm phần tử: 4](#_Toc200310648)

[c) Xóa phần tử: 4](#_Toc200310649)

[d) Duyệt cây: 5](#_Toc200310650)

[**2.** **Cài đặt các thao tác cơ bản của Binary Search Tree trên Python:** 5](#_Toc200310651)

[2.1. Cài đặt Binary Search Tree: 5](#_Toc200310652)

[2.2. Thêm node: 5](#_Toc200310653)

[2.3. Tìm kiếm: 5](#_Toc200310654)

[2.4. Xóa node: 6](#_Toc200310655)

[**3.** **Các phương pháp duyệt cây:** 6](#_Toc200310656)

[3.1. Duyệt in-order (Left-Root-Right): 6](#_Toc200310657)

[3.2. Duyệt pre-order (Root-Left-Right): 7](#_Toc200310658)

[3.3. Duyệt post-order (Left-Right-Root): 7](#_Toc200310659)

[**CHƯƠNG II: SO SÁNH HIỆU SUẤT VỚI DANH SÁCH LIÊN KẾT ĐƠN, VỚI CÂY AVL HOẶC VỚI RED-BLACK TREE.** 7](#_Toc200310660)

[**1.** **So sánh hiệu suất:** 7](#_Toc200310661)

[**2.** **Ký hiệu độ phức tạp:** 8](#_Toc200310662)

[**CHƯƠNG III: ỨNG DỤNG CẤU TRÚC DỮ LIỆU BINARY SEARCH TREE ĐỂ VIẾT CHƯƠNG TRÌNH QUẢN LÝ HỌC PHẦN TẠI HUIT TRÊN NGÔN NGỮ C/C++.** 8](#_Toc200310663)

[**1.** **Phân tích đề bài:** 8](#_Toc200310664)

[**2.** **Phân tích dữ liệu cần lưu:** 9](#_Toc200310665)

[**3.** **Các chức năng cần xây dựng:** 9](#_Toc200310666)

[**4.** **Xây dựng cấu trúc:** 10](#_Toc200310667)

[**5.** **Mô tả chi tiết các chức năng:** 10](#_Toc200310668)

[a) Tạo cây bằng cách nhập từ bàn phím. 10](#_Toc200310669)

[b) Tạo cây bằng cách đọc từ File. 11](#_Toc200310670)

[c) Duyệt cây theo 6 cách. 12](#_Toc200310671)

[d) Duyệt cây theo chiều rộng và chiều sâu. 12](#_Toc200310672)

[e) Xóa một học phần có mã học phần X. 14](#_Toc200310673)

[f) Thêm học phần mới vào cây. 15](#_Toc200310674)

[g) Tìm học phần theo tên. 15](#_Toc200310675)

[h) Thống kê các học phần theo số tín chỉ. 15](#_Toc200310676)

[i) Đếm số học phần theo từng loại. 15](#_Toc200310677)

[j) Tính tổng số tín chỉ. 16](#_Toc200310678)

[**CHƯƠNG IV: KẾT LUẬN** 17](#_Toc200310679)

[**TÀI LIỆU THAM KHẢO** 18](#_Toc200310680)

# **CHƯƠNG I: TỔNG QUAN**

## **Binary Search Tree trong Python:**

1. Khái niệm:

Cây tìm kiếm nhị phân là một cấu trúc dữ liệu được sử dụng trong khoa học máy tính để tổ chức và lưu trữ dữ liệu theo cách được sắp xếp. Mỗi nút trong Cây tìm kiếm nhị phân có nhiều nhất hai nút con, một nút con trái và một nút con phải.

**Tính chất :**

* Giá trị của tất cả nút con trái < nút gốc.
* Giá trị của tất cả nút con phải > nút gốc.

1. Các thao tác cơ bản của Binary Search Tree:
   1. Chèn các phần tử vào cây:

* **Mục đích**: Thêm một phần tử mới vào BST sao cho **cây vẫn duy trì tính chất nhị phân.**
* **Cách thực hiện**:
* Nếu **cây rỗng**, tạo nút mới làm gốc.
* Nếu giá trị **nhỏ hơn** gốc, chèn vào **cây con bên trái**.
* Nếu giá trị **lớn hơn** gốc, chèn vào **cây con bên phải**.
* Dùng **đệ quy** để tìm vị trí thích hợp.
  1. Tìm kiếm phần tử:
* **Mục đích**: Xác định xem một phần tử có tồn tại trong cây hay không.
* **Cách thực hiện**:
* Nếu **gốc bằng giá trị cần tìm**, trả về nút đó.
* Nếu giá trị **nhỏ hơn** gốc, tìm tiếp ở **cây con bên trái.**
* Nếu giá trị **lớn hơn** gốc, tìm tiếp ở **cây con bên phải**.
* Nếu đi đến nút rỗng (NULL), tức là phần tử không tồn tại.
  1. Xóa phần tử:
* **Mục đích**: Loại bỏ một phần tử khỏi cây nhưng vẫn **giữ nguyên tính chất BST.**
* **Các trường hợp cần xử lý**:
* **Node là lá (không có con)** → Xóa trực tiếp.
* **Node có 1 con** → Nối con của nó với cha.
* **Node có 2 con** → Tìm **node nhỏ nhất bên phải**, thay thế, rồi xóa node đó.
  1. Duyệt cây:
* **Mục đích**: In ra các phần tử theo **thứ tự khác nhau**.
* **Các cách duyệt cây**:
* **In-order (Trái - Gốc - Phải).**
* **Pre-order (Gốc - Trái - Phải).**
* **Post-order (Trái - Phải - Gốc).**

## **Cài đặt các thao tác cơ bản của Binary Search Tree trên Python:**

1. Cài đặt Binary Search Tree:

class TreeNode:

def \_\_init\_\_(self, key):

self.key = key

self.left = None

self.right = None

class BinarySearchTree:

def \_\_init\_\_(self):

seft.root = None

1. Thêm node:

def insert(seft, key):

self.root = self.\_insert(self.root, key)

def \_insert(self, node, key):

if node is None:

return TreeNode(key)

if key < node.key:

node.left = self.\_insert(node.left, key)

elif key > node.key:

node.right = self.\_insert(node.right, key)

return node

1. Tìm kiếm:

def search(self, key):

return self.\_search(self.root, key)

def \_search(self, node, key):

if node is None or node.key == key:

return node

if key < node.key:

return self.\_search(node.left, key)

return self.\_search(node.right, key)

1. Xóa node:

def delete(self, key):

self.root = self.\_delete(self.root, key)

def \_delete(self, node, key):

if node is None:

return node

if key < node.key:

node.left = self.\_delete(node.left, key)

elif key > node.key:

node.right = self.\_delete(node.right, key)

else:

if node.left is None:

return node.right

elif node.right is None:

return node.left

temp = self.\_min\_value\_node(node.right)

node.key = temp.key

node.right = self.\_delete(node.right, temp.key)

return node

def \_min\_value\_node(self, node):

current = node

while current.left is not None:

current = current.left

return current

1. **Các phương pháp duyệt cây:**
2. Duyệt in-order (Left-Root-Right):

def in\_order\_traversal(self):

result = []

self.\_in\_order(self.root, result)

return result

def \_in-order(self, node, result):

if node:

self.\_in\_order(node.left, result)

result.append(node.key)

self.\_in\_order(node.right, result)

1. Duyệt pre-order (Root-Left-Right):

def pre\_order\_traversal(self):

result = []

self.\_pre\_order(self.root, result)

return result

def \_pre\_order(self, node, result):

if node:

result.append(node.key)

self.\_pre\_order(node.left, result)

self.\_pre\_order(node.right, result)

1. Duyệt post-order (Left-Right-Root):

def post\_order\_traversal(self):

result = []

self.\_post\_order(self.root, result)

return result

def post\_order(self, node, result):

if node:

self.\_post\_order(node.left, result)

self.\_post\_order(node.right, result)

result.append(node.key)

# **CHƯƠNG II: SO SÁNH HIỆU SUẤT VỚI DANH SÁCH LIÊN KẾT ĐƠN, VỚI CÂY AVL HOẶC VỚI RED-BLACK TREE.**

## **So sánh hiệu suất:**

| **Tiêu chí** | **List** | **BST (Binary Search Tree)** | **AVL Tree** | **Red-Black Tree** |
| --- | --- | --- | --- | --- |
| **Tìm kiếm (Search)** | O(n) | * Tốt: O(log n) * Xấu: O(n) | O(log n) | O(log n) |
| **Thêm phần tử (Insert)** | * O(1) (đầu, cuối) * O(n) (giữa) | * Tốt: O(log n) * Xấu: O(n) | O(log n) | O(log n) |
| **Xóa phần tử (Delete)** | O(n) | * Tốt: O(log n) * Xấu: O(n) | O(log n) | O(log n) |
| **Khả năng cân bằng** | Không | Không | Có (chặt chẽ) | Có (ổn định) |
| **Độ phức tạp khi cài đặt** | Dễ | Dễ | Trung bình – Khó | Khó hơn BST, dễ hơn AVL |

## **Ký hiệu độ phức tạp:**

|  |  |
| --- | --- |
| **Ký hiệu** | **Tốc độ (n càng lớn)** |
| O(1) | Rất nhanh |
| O(log n) | Nhanh |
| O(n) | Trung bình |

# **CHƯƠNG III: ỨNG DỤNG CẤU TRÚC DỮ LIỆU BINARY SEARCH TREE ĐỂ VIẾT CHƯƠNG TRÌNH QUẢN LÝ HỌC PHẦN TẠI HUIT TRÊN NGÔN NGỮ C/C++.**

## **Phân tích đề bài:**

* Tạo cấu trúc dữ liệu để lưu thông tin mỗi học phần như: Mã học phần, tên học phần, loại học phần, số tín chỉ, hệ đào tạo.
* Tạo ra các chức năng chính cho chương trình:
* Thêm học phần mới vào cây.
* Tìm kiếm học phần theo tên học phần.
* Xóa học phần theo mã học phần.
* Duyệt và hiển thị danh sách học phần.

## **Phân tích dữ liệu cần lưu:**

Bảng 3.1. Phân tích dữ liệu cần lưu

|  |  |
| --- | --- |
| **Trường dữ liệu** | **Kiểu dữ liệu** |
| Mã học phần | Char |
| Tên học phần | Char |
| Loại học phần | Char |
| Số tín chỉ | Int |
| Hệ đào tào | Char |

## **Các chức năng cần xây dựng:**

Bảng 3.2. Các chức năng cần xây dựng

|  |
| --- |
| **Chức năng** |
| Tạo cây bằng cách nhập từ bàn phím |
| Tạo cây bằng cách đọc từ File |
| Duyệt cây theo 6 cách |
| Duyệt cây theo chiều rộng và chiều sâu |
| Xóa 1 học phần có mã học phần X khỏi cây |
| Thêm 1 học phần X vào cây |
| Tìm học phần có tên X |
| Thống kê các học phần theo số tín chỉ |
| Đếm số lượng học phần theo từng loại |
| Tính tổng số tín chỉ các học phần |
|  | | |

1. **Xây dựng cấu trúc:**

|  |
| --- |
| struct HocPhan {  char MaHP[11];  char TenHP[41];  char LoaiHP[11];  int SoTC;  char HeDaoTao[21];  };  typedef HocPhan ItemType;  struct BSTNode {  ItemType Info;  BSTNode\* Left;  BSTNode\* Right;  };  struct BSTree  {  BSTNode\* Root;  }; |

1. **Mô tả chi tiết các chức năng:**
2. Tạo cây bằng cách nhập từ bàn phím.

* Người dùng nhập thông tin từng học phần, sau đó các học phần được chèn vào BST dựa trên **Mã học phần.**
* Hàm chèn:

|  |
| --- |
| int insertBSTNode(BSTNode\*& root, BSTNode\* p) {  if (p == NULL) {  return 0;  }  if (root == NULL) {  root = p;  return 1;  }  int soSanh = strcmp(p->Info.MaHP, root->Info.MaHP);  if (soSanh == 0) {  return 0;  }  else if (soSanh < 0) {  insertBSTNode(root->Left, p);  }  else  insertBSTNode(root->Right, p);  } |

* Hàm nhập từ bàn phím:

|  |
| --- |
| void createBSTree\_ByHand(BSTree& bst) {  ItemType x;  int n;  printf("Nhap so luong hoc phan: ");  scanf("%d", &n);  while (getchar() != '\n');  for (int i = 0;i < n;i++) {  printf("\n----------------Nhap hoc phan thu %d----------------\n", i + 1);  nhapThongTinHP(x);  insertBSTNode(bst.Root, createBSTNode(x));  }  } |

1. Tạo cây bằng cách đọc từ File.

* Chương trình mở File chứa thông tin học phần, sau đó đọc từng dòng (Dòng có định dạng: MaHP,TenHP,LoaiHP,SoTC,HeDaoTao).
* Hàm đọc từ File:

|  |
| --- |
| void createBSTree\_FromTextFile(BSTree& bst, char inputFileName[]) {  FILE\* fi = fopen(inputFileName, "r");  if (fi == NULL) {  printf("Loi mo file: %s\n", inputFileName);  return;  }  int n;  fscanf(fi, "%d\n", &n);  int i = 0;  while (i < n) {  ItemType x;  fscanf(fi, "%[^,],%[^,],%[^,],%d,%[^\n]\n", x.MaHP, x.TenHP, x.LoaiHP, &x.SoTC, x.HeDaoTao);  BSTNode\* p = createBSTNode(x);  int kq = insertBSTNode(bst.Root, p);  if (kq == 1) i++;  }  fclose(fi);  } |

1. Duyệt cây theo 6 cách.

* Tạo bảng menu chọn các kiểu để duyệt.
* Hiển thị danh sách học phần theo các kiểu duyệt khác nhau:
  + NLR (Tiền tự trái)
  + NRL (Hậu tự phải)
  + LNR (Trung tự trái)
  + LRN (Hậu tự trái)
  + RNL (Trung tự trái)
  + RLN (Tiền tự phải)
* Mỗi kiểu duyệt sử dụng đệ quy và in ra thông tin học phần.
* Ví dụ về kiểu NLR (Tiền tự trái):

|  |
| --- |
| void traverse\_NLR(BSTNode\* root) {  if (!root) {  return;  }  showBSTNode(root);//Xử lý nút hiên tại  traverse\_NRL(root->Left);//Duyệt cây trái  traverse\_NRL(root->Right);//Duyệt cây phải  } |

1. Duyệt cây theo chiều rộng và chiều sâu.

* Duyệt cây theo chiều sâu:
  + Bước làm:
    1. Khởi tạo stack.
    2. Đưa root vào stack.
    3. Lặp lại cho đến khi stack rỗng:
       - Lấy nút đầu của stack ra.
       - Xuất thông tin nút vừa lấy.
       - Kiểm tra nếu nút trái của nút vừa lấy khác NULL thì đưa vào stack.
       - Kiểm tra nếu nút phải của nút vừa lấy khác NULL thì đưa vào stack.
  + Hàm duyệt theo chiều sâu:

|  |
| --- |
| void Traversal\_depth(BSTNode\* root) {  if (!root) return;  stack<BSTNode\*> st;  st.push(root);  while (!st.empty()) {  BSTNode\* p = st.top();  st.pop();  showBSTNode(p);  // Nếu p có con trái thì thêm vào hàng đợi st  if (p->Left != NULL) {  st.push(p->Left);  }  // Nếu p có con phải thì thêm vào hàng đợi st  if (p->Right != NULL) {  st.push(p->Right);  }  }  } |

* Duyệt cây theo chiều rộng (Duyệt theo tầng):
  + Bước làm:
    1. Khởi tạo queue.
    2. Đưa root vào queue.
    3. Lặp lại cho đến khi queue rỗng:
       - Lấy nút đầu của queue ra.
       - Xuất thông tin nút vừa lấy.
       - Kiểm tra nếu nút trái của nút vừa lấy khác NULL thì đưa vào queue.
       - Kiểm tra nếu nút phải của nút vừa lấy khác NULL thì đưa vào queue.
  + Hàm duyệt theo chiều rộng:

|  |
| --- |
| void Traversal\_width(BSTNode\* root) {  if (!root) return;  queue<BSTNode\*> q;  q.push(root);  while (!q.empty()) {  BSTNode\* tmp = q.front();  q.pop();  showBSTNode(tmp);  // Nếu tmp có con trái thì thêm vào hàng đợi q  if (tmp->Left != NULL) {  q.push(tmp->Left);  }  // Nếu tmp có con phải thì thêm vào hàng đợi q  if (tmp->Right != NULL) {  q.push(tmp->Right);  }  }  } |

1. Xóa một học phần có mã học phần X.

* So sánh MaHP cần xóa với từng nút của cây để xác định đi trái hay phải.
* Khi tìm thấy nút cần xóa:
  + Trường hợp 1: Nút không có con trái 🡪 nối con phải lên thay.
  + Trường hợp 2: Nút không có con phải 🡪 nối con trái lên thay.
  + Trường hợp 3: Nút có 2 con 🡪 tìm nút thế mạng **p** là nút trái nhất của cây con bên phải 🡪 copy dữ liệu của **p** vào nút cần xóa 🡪 xóa nút p.
* Hàm tìm nút thế mạng

|  |
| --- |
| BSTNode\* findBSTNodeReplace(BSTNode\*& p) {  BSTNode\* f = p;  BSTNode\* q = p->Right;  while (q->Left != NULL) {  f = q;  q = q->Left;  }  p->Info = q->Info;  if (f == p)  f->Right = q->Right;  else  f->Left = q->Right;  return q;  } |

* Hàm xóa một học phần có mã học phần X.

|  |
| --- |
| int deleteBSTNode\_MaHP(BSTNode\*& root, char MaHP[]) {  if (root == NULL) //Cây rỗng  return 0;  if (strcmpi(root->Info.MaHP, MaHP) > 0)  return deleteBSTNode\_MaHP(root->Left, MaHP);  else if (strcmpi(root->Info.MaHP, MaHP) < 0)  return deleteBSTNode\_MaHP(root->Right, MaHP);  else{  BSTNode\* p = root;  if (root->Left == NULL) {  root = root->Right;  delete p;  }  else if (root->Right == NULL) {  root = root->Left;  delete p;  }  else {  BSTNode\* q = findBSTNodeReplace(p);  delete q;  }  }  return 1;  } |

1. Thêm học phần mới vào cây.

* Nhập thông tin học phần sau đó gọi hàm **insertBSTNode** để thêm học phần mới vào cây.

1. Tìm học phần theo tên.

|  |
| --- |
| BSTNode\* findBSTNode\_TenHP(BSTNode\* root, char tenHP[]) {  if (root == NULL) return NULL;  if (strcmpi(root->Info.TenHP, tenHP) == 0) return root;  BSTNode\* findLeft = findBSTNode\_TenHP(root->Left, tenHP);  if (findLeft != NULL) return findLeft;  return findBSTNode\_TenHP(root->Right, tenHP);  } |

1. Thống kê các học phần theo số tín chỉ.

|  |
| --- |
| void Statistic(BSTNode\* root,int soTC) {  if (!root)  {  return;  }  Statistic(root->Left, soTC);  if (root->Info.SoTC == soTC)  {  showBSTNode(root);  }  Statistic(root->Right, soTC);  } |

1. Đếm số học phần theo từng loại.

|  |
| --- |
| int countCourseType(BSTNode\* root,char loaiHP[]) {  if (!root)  {  return 0;  }  int countL = countCourseType(root->Left, loaiHP);  int countR = countCourseType(root->Right, loaiHP);  if (strcmpi(root->Info.LoaiHP, loaiHP) == 0) return countL + countR + 1;  return countL + countR;  } |

1. Tính tổng số tín chỉ.

|  |
| --- |
| int sumTinChi(BSTNode\* root) {  if (!root)  {  return 0;  }  int sumL = sumTinChi(root->Left);  int sumR = sumTinChi(root->Right);  return sumL + sumR + root->Info.SoTC;  } |

# **CHƯƠNG IV: KẾT LUẬN**

Trong quá trình thực hiện đồ án “Nghiên cứu Cấu trúc Dữ liệu Cây Nhị Phân Tìm Kiếm trong Python - Ứng dụng cấu trúc dữ liệu Binary Search Tree để viết chương trình bằng ngôn ngữ C/C++ hãy thực hiện Bài toán quản lý học phần tại HUIT”, nhóm đã củng cố được kiến thức về cấu trúc dữ liệu, đặc biệt là cây nhị phân tìm kiếm (BST) và vận dụng nó để giải quyết bài toán thực tế. Qua đó, nhóm cũng rèn luyện được kỹ năng làm việc nhóm, tư duy thuật toán và kỹ năng lập trình với ngôn ngữ C/C++.

Mặc dù đã cố gắng hoàn thành tốt nhất trong khả năng, nhưng do thời gian và kinh nghiệm còn hạn chế, chắc chắn đồ án vẫn còn một số thiếu sót. Nhóm rất mong nhận được sự góp ý từ thầy/cô để có thể cải thiện và phát triển tốt hơn trong những dự án sau.

# **TÀI LIỆU THAM KHẢO**

[1]. Trần Hạnh Nhi và Dương Anh Đức, Giáo trình cấu trúc dữ liệu và giải thuật, Đại học Quốc gia TP.HCM, 2001.

[2]. Nguyễn Hồng Chương, Cấu trúc dữ liệu - Ứng dụng và cài đặt bằng C, NXB TP.HCM, 2003.

[3]. Nguyễn Quốc Cường – Hoàng Đức Hải, Cấu trúc dữ liệu + Giải thuật = Chương trình, NXB Giáo dục, 1996.

[4]. Nguyễn Trung Trực, Cấu trúc dữ liệu, Đại học Bách khoa TP.HCM, 1994.

[5]. Nhóm tác giả Khoa CNTT, Cấu trúc dữ liệu và giải thuật, Trường Cao đẳng Công nghiệp Thực phẩm TP.HCM, 2007.