**TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP.HCM**

**KHOA CÔNG NGHỆ THÔNG TIN**

****

**ĐỒ ÁN CUỐI KÌ   
MÔN CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT**

**TÌM HIỂU K-D TREE VÀ ỨNG DỤNG MINH HỌA**

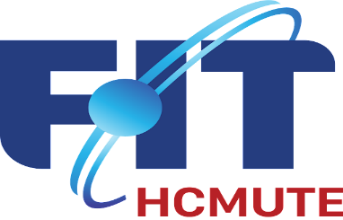
**Nhóm sinh viên thực hiện:**

1. **Nguyễn Kiều Châu Anh – 20110234**
2. **Lâm Hoàng Duyên - 20110174**
3. **Nguyễn Thị Quỳnh Như – 20110235**
4. **Phan Thị Diễm Trinh – 20110193**

**Ngành: CÔNG NGHỆ THÔNG TIN**

**GVHD: Huỳnh Xuân Phụng**

**Tp. Hồ Chí Minh, tháng 12 năm 2021**

**CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM**

**Độc lập – Tự do – Hạnh phúc**

---\*\*\*---

## PHIẾU NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

Họ và tên sinh viên: NGUYỄN KIỀU CHÂU ANH – MSSV: 20110234

Họ và tên sinh viên: LÂM HOÀNG DUYÊN – MSSV: 20110174

Họ và tên sinh viên: NGUYỄN THỊ QUỲNH NHƯ – MSSV: 20110235

Họ và tên sinh viên: PHAN THỊ DIỄM TRINH – MSSV: 20110193

Ngành: Công nghệ thông tin

Tên đề tài: Tìm hiểu K-D tree và ứng dụng minh họa

Họ và tên Giáo viên hướng dẫn: Huỳnh Xuân Phụng

**NHẬN XÉT**

1. Về nội dung đề tài & khối lượng thực hiện:

2. Ưu điểm:

3. Khuyết điểm:

4. Đề nghị cho bảo vệ hay không?

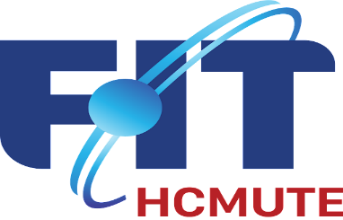
5. Đánh giá loại:

6. Điểm: (Bằng chữ: )

Tp. *Hồ Chí Minh*, ngày tháng năm 20…

Giáo viên hướng dẫn

*(Ký & ghi rõ họ tên)*

**CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM**

**Độc lập – Tự do – Hạnh phúc**

---\*\*\*---

**PHIẾU NHẬN XÉT CỦA GIÁO VIÊN PHẢN BIỆN**

Họ và tên sinh viên: NGUYỄN KIỀU CHÂU ANH – MSSV: 20110234

Họ và tên sinh viên: LÂM HOÀNG DUYÊN – MSSV: 20110174

Họ và tên sinh viên: NGUYỄN THỊ QUỲNH NHƯ – MSSV: 20110235

Họ và tên sinh viên: PHAN THỊ DIỄM TRINH – MSSV: 20110193

Ngành: Công nghệ thông tin

Tên đề tài: Tìm hiểu K-D tree và ứng dụng minh họa

Họ và tên Giáo viên phản biện: Huỳnh Xuân Phụng

**NHẬN XÉT**

1. Về nội dung đề tài & khối lượng thực hiện:

2. Ưu điểm:

3. Khuyết điểm:

4. Đề nghị cho bảo vệ hay không?

5. Đánh giá loại:

6. Điểm: (Bằng chữ: )

Tp. *Hồ Chí Minh*, ngày tháng năm 20…

Giáo viên phản biện

*(Ký & ghi rõ họ tên)*

**LỜI CẢM ƠN**

Nhóm em xin phép gửi lời cảm ơn chân thành đến thầy Huỳnh Xuân Phụng, người đã trực tiếp hỗ trợ nhóm em trong suốt quá trình định chọn đề tài, hướng dẫn, nhận xét và góp ý cũng như cung cấp những lời khuyên giúp chúng em thực hiện đồ án tốt nhất. Nhờ có những chỉ dẫn của thầy đã giúp nhóm em hiểu rõ về kiến thức để thực hiện, các cách trình bày cũng như thực hiện đồ án nên chúng em đã hoàn thành đúng tiến độ cùng với đó là rất nhiều kinh nghiệm được chúng em học hỏi. Một lần nữa, nhóm em xin cảm ơn thầy.

Em cũng xin chân thành cảm ơn các quý thầy cô trong khoa Công Nghệ Thông Tin đã giúp đỡ hỗ trợ kiến thức cũng như giải đáp thắc mắc của em. Cùng với đó, em xin được gửi cảm ơn đến các bạn cùng khóa đã cung cấp nhiều thông tin và kiến thức hữu ích giúp cho em hoàn thiện đề tài hơn.

Đồ án được thực hiện trong vòng tám tuần, khoảng thời gian cũng vừa đủ để nhóm em hoàn tất được đồ án. Tuy nhiên do nhiều kiến thức mới cũng như thời gian chúng em thực hiện qua từng tuần không phải là tối ưu nhất nên đồ án thực hiện sẽ có nhiều sai sót, đó là điều không thể tránh khỏi. Chúng em rất mong nhận được mọi ý kiến đóng góp qu‎ý báu của các thầy cô để giúp cho kiến thức của chúng em ngày càng vững chắc hơn. Chúng em xin chân thành cảm ơn.

Thành phố Hồ Chí Minh, ngày 17 tháng 12 năm 2021

## MỤC LỤC

[PHIẾU NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN i](#_Toc91870105)

[PHIẾU NHẬN XÉT CỦA GIÁO VIÊN PHẢN BIỆN ii](#_Toc91870106)

[LỜI CẢM ƠN iii](#_Toc91870107)

[MỤC LỤC iv](#_Toc91870108)

[DANH SÁCH CÁC HÌNH ẢNH vi](#_Toc91870109)

[DANH SÁCH CÁC BẢNG BIỂU vii](#_Toc91870110)

[CHƯƠNG 1: GIỚI THIỆU 1](#_Toc91870111)

[1.1. GIỚI THIỆU VỀ ĐỀ TÀI 1](#_Toc91870112)

[1.2. MỤC TIÊU, LÝ DO CHỌN ĐỀ TÀI 1](#_Toc91870113)

[1.3. CÔNG CỤ THỰC HIỆN 1](#_Toc91870114)

[1.4. PHẠM VI NGHIÊN CỨU 1](#_Toc91870115)

[CHƯƠNG 2: CƠ SỞ LÝ THUYẾT 2](#_Toc91870116)

[2.1. ĐỊNH NGHĨA K-D TREE 2](#_Toc91870117)

[2.2. PHÂN TÍCH HIỆU SUẤT, ĐỘ PHỨC TẠP CỦA K-D TREE 5](#_Toc91870118)

[2.2.1. K-D Tree tối ưu qua phương pháp tìm trung vị 5](#_Toc91870119)

[2.2.2. Phân tích 5](#_Toc91870120)

[2.3. HẠN CHẾ CỦA K-D TREE 6](#_Toc91870121)

[CHƯƠNG 3: HIỆN THỰC K-D TREE VÀ ỨNG DỤNG 7](#_Toc91870122)

[3.1. HIỆN THỰC THUẬT TOÁN – DỰNG K-D TREE 7](#_Toc91870123)

[3.2. CÁC THAO TÁC CƠ BẢN VỚI K-D TREE 12](#_Toc91870124)

[3.2.1. Tìm node nhỏ nhất 12](#_Toc91870125)

[3.2.2. Xóa một node ra khỏi K-D tree 14](#_Toc91870126)

[3.2.3. In K-D tree 2 chiều, 3 chiều 19](#_Toc91870127)

[3.3 ỨNG DỤNG K-D TREE 20](#_Toc91870128)

[3.3.1. Ứng dụng K-D tree trong thuật toán tìm kiếm K-Nearest Neighbor 20](#_Toc91870129)

[3.3.1.1 Thuật toán tìm kiếm K - Nearest Neighbor trên K-D tree 21](#_Toc91870130)

[3.3.1.2 Thuật toán tìm kiếm Naive K - Nearest Neighbor 23](#_Toc91870131)

[3.3.1.3 So sánh thuật toán Naive K - Nearest Neighbor và thuật toán K - Nearest Neighbor trên K-D tree 25](#_Toc91870132)

[3.3.2. Ứng dụng thuật toán tìm kiếm K-Nearest Neighbor trên K-D tree trong bài toán cụ thể 26](#_Toc91870133)

[CHƯƠNG 4: KẾT LUẬN 32](#_Toc91870134)

[4.1. KẾ HOẠCH THỰC HIỆN 32](#_Toc91870135)

[4.2. KẾT QUẢ ĐẠT ĐƯỢC 32](#_Toc91870136)

[4.3. KHÓ KHĂN GẶP PHẢI 32](#_Toc91870137)

[4.3.1. Công nghệ 32](#_Toc91870138)

[4.3.2. Quá trình thực hiện 32](#_Toc91870139)

[4.4. Kinh nghiệm đạt được 33](#_Toc91870140)

[TÀI LIỆU THAM KHẢO 34](#_Toc91870141)

[LINK GITHUB CỦA ĐỒ ÁN 35](#_Toc91870142)

## DANH SÁCH CÁC HÌNH ẢNH

[Hình 2.1 Ví dụ minh họa xây dựng K-D tree (1) 3](#_Toc91534471)

[Hình 2.2 Ví dụ minh họa xây dựng K-D tree (2) 3](#_Toc91534472)

[Hình 2.3 Ví dụ minh họa xây dựng K-D tree (3) 4](#_Toc91534473)

[Hình 2.4 Ví dụ minh họa xây dựng K-D tree (4) 4](#_Toc91534474)

[Hình 3.1 Struct của K-D tree 7](#_Toc91869943)

[Hình 3.2 Thuật toán tìm trung vị 9](#_Toc91869944)

[Hình 3.3 Thuật toán dựng cây 10](#_Toc91869945)

[Hình 3.4 Thời gian chạy của thuật toán dựng K-D tree 11](#_Toc91869946)

[Hình 3.5 So sánh thời gian chạy của thuật toán dựng K-D tree 2 chiều và 3 chiều 12](#_Toc91869947)

[Hình 3.6 Thuật toán tìm node nhỏ nhất trong 3 node 13](#_Toc91869948)

[Hình 3.7 Thuật toán tìm node nhỏ nhất theo nhánh 14](#_Toc91869949)

[Hình 3.8 Thuật toán xóa một node ra khỏi K-D tree 17](#_Toc91869950)

[Hình 3.9 Thời gian chạy của thuật toán xóa trong K-D tree 2 chiều 18](#_Toc91869951)

[Hình 3.10 So sánh thời gian chạy của thuật toán xóa trong K-D tree 19](#_Toc91869952)

[Hình 3.11 Thuật toán in K-D tree 20](#_Toc91869953)

[Hình 3.12 Thuật toán tìm nearest neighbor 22](#_Toc91869954)

[Hình 3.13 Thời gian chạy của thuật toán tìm kiếm K - Nearest Neighbor trên K-D tree (2 chiều) 23](#_Toc91869955)

[Hình 3.14 Thuật toán tìm kiếm Naive K - Nearest Neighbor 24](#_Toc91869956)

[Hình 3.15 Thời gian chạy của thuật toán Naive K - Nearest Neighbor 25](#_Toc91869957)

[Hình 3.16 So sánh thời gian chạy của thuật toán Naive K - Nearest Neighbor và thuật toán K - Nearest Neighbor trên K-D tree (2 chiều) 26](#_Toc91869958)

[Hình 3.17 Tạo kiểu cấu trúc để in màu 27](#_Toc91869959)

[Hình 3.18 Đưa dữ liệu màu vào mảng cấu trúc Colour\_code\_name input[] 28](#_Toc91869960)

[Hình 3.19 Đưa dữ liệu tên màu vào mảng cấu trúc Colour\_code\_name input[] 29](#_Toc91869961)

[Hình 3.20 Dựng KD-Tree từ file chứa các mã màu 30](#_Toc91869962)

[Hình 3.21 Áp dụng thuật toán nearest neighbors search để tìm ra mã màu 31](#_Toc91869963)

[Hình 3.22 So sánh mã màu tìm được để tìm ra màu cần tìm kiếm 31](#_Toc91869964)

**DANH SÁCH CÁC BẢNG BIỂU**

[Bảng 3.1 Thời gian chạy của thuật toán dựng K-D tree 10](#_Toc91870008)

[Bảng 3.2 So sánh thời gian chạy của thuật toán dựng K-D tree 2 chiều và 3 chiều 11](#_Toc91870009)

[Bảng 3.3 Thời gian chạy của thuật toán xóa trong K-D tree 2 chiều 18](#_Toc91870010)

[Bảng 3.4 So sánh thời gian chạy của thuật toán xóa trong K-D tree 19](#_Toc91870011)

[Bảng 3.5 In trong K-D tree 2 chiều và K-D tree 3 chiều 20](#_Toc91870012)

[Bảng 3.6 Thời gian chạy của thuật toán tìm kiếm K - Nearest Neighbor trên K-D tree   
(2 chiều) 22](#_Toc91870013)

[Bảng 3.7 Thời gian chạy của thuật toán Naive K - Nearest Neighbor 24](#_Toc91870014)

[Bảng 3.8 So sánh thời gian chạy của thuật toán Naive K - Nearest Neighbor và thuật toán K - Nearest Neighbor trên K-D tree (2 chiều) 25](#_Toc91870015)

[Bảng 4.1 Kế hoạch thực hiện đồ án 29](#_Toc91534620)

[Bảng 4.2 Khó khăn về công nghệ 29](#_Toc91534621)

[Bảng 4.3 Khó khăn trong quá trình thực hiện 30](#_Toc91534622)

# CHƯƠNG 1: GIỚI THIỆU

* 1. **GIỚI THIỆU VỀ ĐỀ TÀI**

Các đối tượng khác nhau trong không gian đều có những tính chất, đặc điểm khác nhau. Chính vì vậy, bài toán liên quan đến khai thác dữ liệu đòi hỏi các lập trình viên một cấu trúc lưu trữ nhằm tổ chức, quản lí, bảo trì dữ liệu lớn sao cho hiệu quả nhất. Có nhiều cấu trúc lưu trữ đã được ra đời để đáp ứng yêu cầu này, trong đó, không thể không nhắc tới   
K-D tree, một phương pháp lưu trữ dữ liệu bằng cách sử dụng các node biểu diễn trong không gian k chiều để phân cụm các dữ liệu liên quan với nhau ở một hoặc vài tính chất, đặc điểm nào đó. K-D tree được áp dụng nhiều trong các lĩnh vực lưu trữ dữ liệu, và là một nội dung quan trọng trong các học phần Cấu trúc dữ liệu và giải thuật ở nhiều trường đại học trên thế giới nói chung, Việt Nam nói riêng, trong đó có trường đại học Sư phạm kỹ thuật thành phố Hồ Chí Minh. Chính vì vậy, dưới sự hướng dẫn của thầy Huỳnh Xuân Phụng, nhóm chúng em quyết định chọn đề tài **K-D tree và ứng dụng của K-D tree** làm đồ án cuối kì.

* 1. **MỤC TIÊU, LÝ DO CHỌN ĐỀ TÀI**

Trong khung thời gian phạm vi thực hiện, bài báo cáo sẽ đề cập tới các nội dung chính sau đây:

• Cơ sở lí thuyết liên quan tới K-D tree: định nghĩa, phương pháp tìm kiếm trên cây, giải thích về balanced K-D tree, so sánh với các thuật toán khác, phân tích hiệu suất, hạn chế của K-D tree.

• Hiện thực thuật toán thông qua ứng dụng đơn giản: Tìm một điểm màu RGB cho trước trong file dữ liệu màu.

* 1. **CÔNG CỤ THỰC HIỆN**

Ngôn ngữ lập trình C/C++.

* 1. **PHẠM VI NGHIÊN CỨU**

- Lí thuyết cơ sở về K-D tree.

- Hiện thực K-D tree.

**CHƯƠNG 2: CƠ SỞ LÝ THUYẾT**

**2.1. ĐỊNH NGHĨA K-D TREE**

Cây nhị phân K chiều (K-D tree hay K-dimentions tree) được phát triển bởi bởi Jon Bentley, và là một biến thể của BST (Binary Search Tree - Cây tìm kiếm nhị phân). K-D tree là một cấu trúc dữ liệu dạng cây được sử dụng để biểu diễn các điểm trong không gian K chiều. Nó được sử dụng cho các ứng dụng khác nhau như điểm gần nhất (trong không gian K chiều), lưu trữ hiệu quả dữ liệu không gian, tìm kiếm phạm vi, v.v.

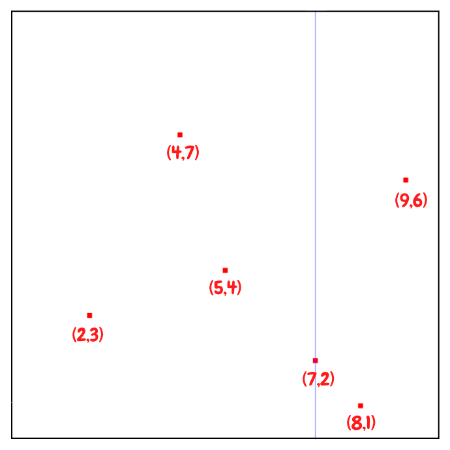
Mỗi nút trong cây đại diện cho một điểm trong không gian. Bất kỳ nút bên trong nào trong cấu trúc này đều chia không gian thành 2 nửa. Nút con bên trái đại diện cho nửa bên trái trong khi nút con bên phải đại diện cho nửa bên phải. Không gian được chia thành 2 nửa không phân biệt số lượng chiều. Nếu là không gian 2 chiều nó là đường thẳng còn với không gian 3 chiều nó là mặt phẳng.

Việc cây hoạt động theo quy tắc như vậy đem lại hiệu quả cho việc tìm kiếm. Việc tìm kiếm ta sẽ dựa vào một số phép phân hạch nhằm loại bỏ các vùng tìm kiếm không cần thiết và rút ngắn được thời gian tìm kiếm.

Để hiểu rõ hơn về thuật toán, cùng xét một ví dụ:

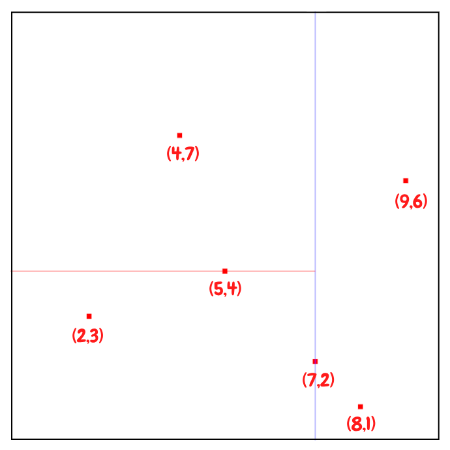
Cho tập hợp E trong không gian 2D tùy ý E = (2,3), (5,4), (9,6), (4,7), (8,1), (7,2). Chúng ta sử dụng thuật toán trung vị để dựng điểm trên thành K-D tree.

Bước 1: Khi xây dựng nút gốc, thứ tự của điểm trên được thiết lập theo thứ nguyên (x) từ nhỏ đến lớn là: (2,3), (4,7), (5,4), (7,2), (8,1), (9,6). Chúng ta sẽ chọn đường trung bình dọc theo trục x làm nút gốc. Theo đó ta chọn (7,2) là nút, điểm được chọn sẽ này chia toàn bộ mặt phẳng thành các phần bên trái và bên phải. Các điểm còn lại so sánh với trung vị nếu x <= 7 là (2,3), (5,4), (4,7), ở phần con bên trái, và điểm (8,1), (9,6) x>7 sẽ nằm bên phải.



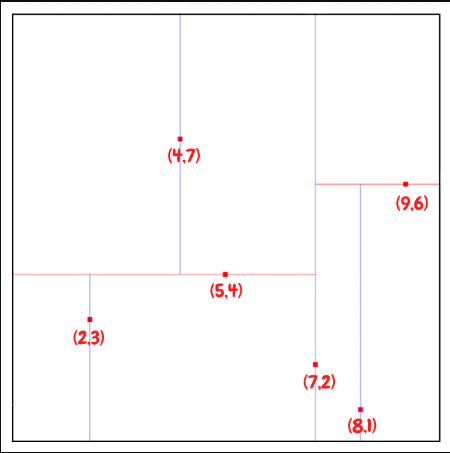
Hình 2.1 Ví dụ minh họa xây dựng K-D tree (1)

Bước 2: Tiếp đó với phần con bên trái chúng ta sẽ chọn (5,4) làm nút, trung bình của ba điểm (2,3), (5,4), (4,7) cho y. Tương tự nhánh bên phải lấy (9,6) làm nút.



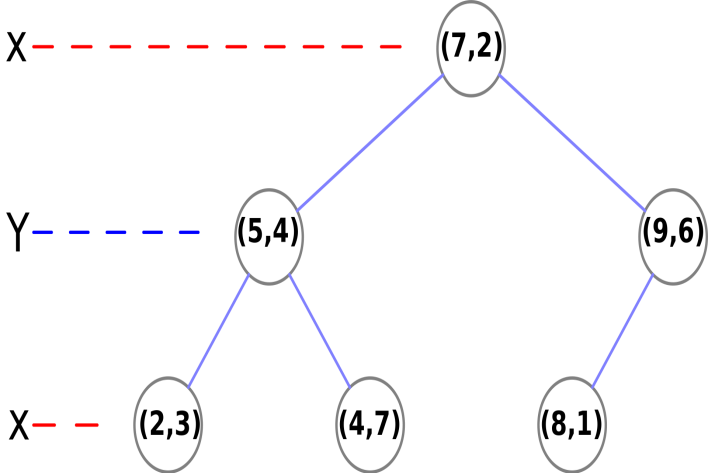
Hình 2.2 Ví dụ minh họa xây dựng K-D tree (2)

Bước 3: Cứ tiếp tục thực hiện đệ quy như vậy với x, y lần lượt đổi vị trí cho nhau đến khi không còn chia được nữa, ta sẽ thu được cây.



Hình 2.3 Ví dụ minh họa xây dựng K-D tree (3)

Thu được kết quả là K-D tree có dạng



Hình 2.4 Ví dụ minh họa xây dựng K-D tree (4)

Với không gian phân hoạch như trên việc tìm kiếm một điểm sẽ trở nên nhanh chóng hơn từ việc sử dụng một số phép toán để phân hoạch loại bỏ những vùng không gian khác nhằm tối ưu thời gian tìm kiếm và sự chính xác được nâng lên cao hơn.

Thuật toán K-D tree là một biến thể của cây nhị phân vì vậy cũng sẽ có những phép toán như cây nhị phân như tìm kiếm, chèn hoặc xóa các nút trong thời gian O(log(n)). Điểm khác biệt của K-D tree là tại mỗi đỉnh chúng lưu trữ một tọa độ trên không gian hai hay nhiều chiều nên chúng ta lưu giá trị chỉ chiều của đỉnh.

**2.2. PHÂN TÍCH HIỆU SUẤT, ĐỘ PHỨC TẠP CỦA K-D TREE**

**2.2.1. K-D Tree tối ưu qua phương pháp tìm trung vị**

Việc tìm kiếm và duyệt node trên K-D tree phụ thuộc chủ yếu vào vị trí của điểm truy vấn. Tuy nhiên, ta thường không biết trước được vị trí của các điểm này, và các vị trí này cũng có thể thay đổi khi cây thay đổi. Chính vì vậy, ta cần một K-D tree tối ưu không bị phụ thuộc vào vị trí của node truy vấn trong khi số node cần duyệt là tối thiểu nhất có thể. Ngoài ra, đó phải là một cây mà thứ nguyên và giá trị tại thứ nguyên đó chỉ phụ thuộc vào node chứa nó; như vậy, cây mới có thể áp dụng duyệt hồi quy được. Một K-D tree tối ưu như đề cập không phải là tối ưu nhất trong một trường hợp cụ thể nào nhưng có thể xem là tối ưu nhất trong tất cả trường hợp. Với hai đều kiện nêu trên, Otair đã đề xuất phương pháp tìm trung vị (median) làm biên phân chia các giá trị của các node, bất kể là đang xét thứ nguyên nào. Trong phương pháp tìm trung vị, các thứ nguyên được lựa chọn sao cho các giá trị được chia đều. Thực tế, phương pháp xác định trung vị là một kĩ thuật rất quan trọng trong việc dựng K-D tree cân đối (balanced K-D tree). Hiệu quả của phương pháp xác định trung vị trong việc dựng cây sẽ được mô tả cụ thể hơn trọng mục phân tích.

**2.2.2. Phân tích**

Theo Mohammed Abdallh Otair, lưu trữ cần cho việc sắp xếp file tỉ lệ thuận với kích thước *N* của file. Thứ nguyên và giá trị ở thứ nguyên của các node sẽ được lưu lại. Số nốt lá là *[]-1*, trong đó, *b* là tổng số node lá của cây. Ở mỗi độ sâu, ta sẽ duyệt từng node một, vì vậy, số tính toán thực hiện tỉ lệ thuận với kN, số tính toán cần thiết để dựng cây từ đầu sẽ là *kNlogN*, với *k* là số chiều của K-D tree.

Về thời gian thực thi, K-D tree thường được hình dung trong mô hình hình học. Nghĩa là, nếu ta đưa *k* giá trị của các node vào trục tương ứng, vậy thì mỗi node sẽ là một điểm biểu diễn trong không gian *k* chiều, kể cả điểm cần truy vấn. Hay nói cách khác, toàn bộ file chính là một tập hợp lữu trữ các điểm biểu diễn được trong không gian *k* chiều. Vì thế, bài toán về “best match” (tìm dữ liệu tương đồng nhất) được đơn giản hóa trở thành tìm kiếm *m* điểm có khoảng cách gần nhất với điểm cần truy vấn trong không gian *k* chiều. Khi áp dụng phương pháp xác định trung vị trong việc dựng K-D tree, Otair đã nhận thấy rằng phương pháp này giúp mỗi vùng không gian con chỉ chứa đúng tối đa *b* node. Việc chọn thứ nguyên sao cho đều cũng giúp cho K-D tree được cân đối, các vùng không gian được chia ra có kích thước xấp xỉ nhau. Otair đã tính toán và chứng minh được rằng, để tối thiểu số node cần duyệt thì mỗi không gian của lá chỉ nên chứa duy nhất một node; và số node cần duyệt hoàn toàn không phụ thuộc vào kích thước của file và hàm phân phối *p(X)* của các không gian con. Xét về ý nghĩa hình học, nếu muốn giảm thiểu số node để duyệt bằng cách giảm thiểu việc xét trùng node do các không gian phân chia bị trùng vào nhau, cách tốt nhất là ta chia không gian nhỏ nhất và chứa ít node nhất có thể.

Khi K-D tree là một cây cân đối (balanced tree), thời gian để duyệt cây từ gốc tới lá sẽ là *logN* và tỉ lệ với kích thước *N* của file. Qua thực nghiệm, Otair chứng minh rằng, số node duyệt khi tìm kiếm các điểm lân cận với một điểm cho trước không phụ thuộc vào *N* và đưa ra kết luận rằng thời gian ước lượng cho việc tìm kiếm các điểm lân cận với một điểm cho trước tỉ lệ với *logN.*

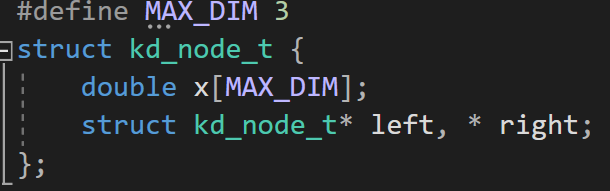
**2.3. HẠN CHẾ CỦA K-D TREE**

K-D tree là một thuật toán thông dụng trong lĩnh vực tìm kiếm. Nhưng các phương pháp thuật toán đều có những mặt không tối ưu và hoàn hảo khi sử dụng và tìm hiểu. Việc hoạt động của cây tương đối đơn giản nhưng việc cấu trúc cây có thể tương đối khó khăn. Trong quá trình hoạt động khi số lượng dữ liệu tìm kiếm tăng lên, cây có thể mang lại hiệu suất kém trong quá trình tìm kiếm. Việc duyệt xét, cắt bỏ trong trường hợp này khá tốn thời gian, các dữ liệu cuối mang một con số gần đúng. Thuật toán K-D khá hiệu quả trong các thuật toán tìm kiếm nhỏ và tương đối, đối với các bài toán phức tạp việc sử dụng thuật toán không mang lại nhiều sự tối ưu cũng như hiệu quả cao cho kết quả sau cùng.

CHƯƠNG 3: HIỆN THỰC K-D TREE VÀ ỨNG DỤNG

3.1. HIỆN THỰC THUẬT TOÁN – DỰNG K-D TREE

Mỗi lá trên K-D tree được tổ chức dưới dạng 1 node và có cấu trúc như sau

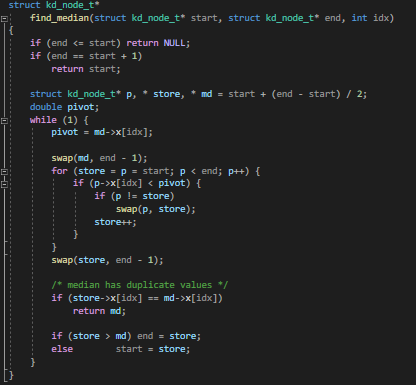
****

Hình 3.1 Struct của K-D tree

Để tạo ra một cây, ta thêm lần lượt các node root vào cây. Tuy nhiên, việc thêm lần lượt hoặc bất kì node sẽ dẫn tới cây sẽ mất cân bằng và làm cho việc truy cập dữ liệu trong cây tốn dữ liệu và kém hiệu quả. Vì vậy cần sử dụng phương pháp chọn trung vị cây (hàm find\_median) để tạo ra K-D tree cân bằng

Thuật toán tìm trung vị của K-D tree với input là vị trí đầu start, cuối end và thứ nguyên i của tập phần tử đang xét; output là node trung vị chính xác. Các bước thực hiện diễn ra như sau:

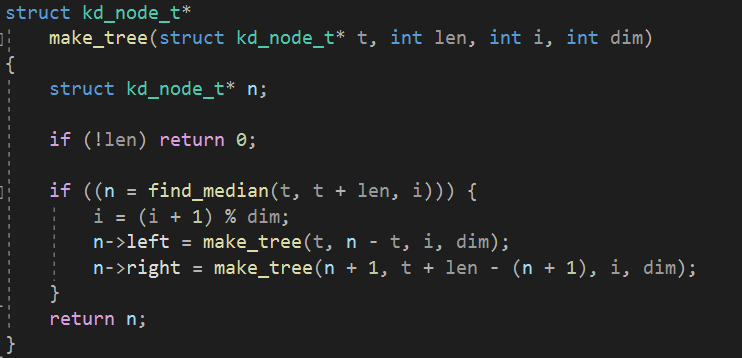
* Bước 1: Kiểm tra node start và node end. Nếu end < start, return null. Nếu end=start+1 thì return start. Ngược lại thực hiện tiếp bước 2.
* Bước 2: Trung vị tạm thời md=start+(end-start)/2.
* Bước 3: Bắt đầu vòng lặp while cho đến khi lệnh return được thực hiện.
* Bước 3.1: pivot=md->x[idx]Gán giá trị tại tọa độ idx (idx có thể là: 0, 1 ,2) của node trung vị md vào pivot.
* Bước 3.2: swap (md, end-1)
* Bước 3.3: Bắt đầu vòng lặp for (p=store=start; p<end; p++)
* Nếu p->x[idx] < pivot, kiểm tra p !=store. Nếu đúng swap (p, store). Ngược lại, store++.
* Bước 3.4: swap (store, end-1)
* Bước 3.5: Nếu store->x[idx] = = md->x[idx] thì return md. Ngược lại thực hiện bước 3.6.
* Bước 3.6: Nếu store>end thì cập nhật end=store. Ngược lại, cập nhật start=store và bắt đầu là bước 3 tới khi khi return trung vị chính xác.



Hình 3.2 Thuật toán tìm trung vị

Thuật toán dựng cây với input là vị trí của phần tử đầu tiên trong mảng truyền vào, chiều dài mảng đang xét len, thứ nguyên i, số chiều dim. Output của thuật toán dựng cây là node gốc của cây cân bằng khi dựng xong theo phương pháp trung vị. Thuật toán diễn ra theo các bước sau:

* Bước 1: Kiểm tra đầu vào, nếu không có dữ liệu vào thì trả về 0.
* Bước 2: Gọi hàm tìm trung vị và trả kết quả vào node n
* Bước 3: Tính lại i = (i+1)%dim, gọi hàm make\_tree hồi quy dựng cây với các node nhánh trái của n
* Bước 4: Tính lại i = (i+1)%dim, gọi hàm make\_tree hồi quy dựng cây với các node nhánh phải của n
* Bước 5: Cây cân bằng hoàn tất, trả về node root n



Hình 3.3 Thuật toán dựng cây

Độ phức tạp về thời gian chạy của thuật toán dựng cây K-D tree:

* Trường hợp tốt nhất và trung bình: O(logN)
* Trường hợp tệ nhất: O(N)

Thời gian chạy của thuật toán dựng K-D tree

|  |  |  |  |
| --- | --- | --- | --- |
| N – phần tử | MIN | MAX | MEAN |
| 10 | 0.000003 | 0.0000052 | 0.000003554 |
| 20 | 0.000007 | 0.000154 | 0.000011489 |
| 30 | 0.00001 | 0.000527 | 0.000012706 |
| 40 | 0.000016 | 0.000916 | 0.00002355 |
| 50 | 0.000022 | 0.001029 | 0.000036742 |
| 60 | 0.000026 | 0.001838 | 0.000036 |
| 70 | 0.000033 | 0.002237 | 0.000056 |
| 80 | 0.000042 | 0.002822 | 0.000081 |
| 90 | 0.000046 | 0.002826 | 0.000084 |
| 100 | 0.000065 | 0.003 | 0.000104 |

Bảng 3.1 Thời gian chạy của thuật toán dựng K-D tree

Hình 3.4 Thời gian chạy của thuật toán dựng K-D tree

So sánh thời gian chạy của thuật toán dựng K-D tree 2 chiều, 3 chiều

|  |  |  |
| --- | --- | --- |
| N – phần tử | K-D tree 2 chiều | K-D tree 3 chiều |
| 10 | 0.000008 | 0.000009 |
| 100 | 0.00008 | 0.0001058 |
| 1000 | 0.001 | 0.002 |
| 10 000 | 0.016 | 0.016 |
| 100 0000 | 0.159 | 0.1758 |

Bảng 3.2 So sánh thời gian chạy của thuật toán dựng K-D tree 2 chiều và 3 chiều

Hình 3.5 So sánh thời gian chạy của thuật toán dựng K-D tree 2 chiều và 3 chiều

Biểu đồ cho thấy thời gian chạy của thuật toán dựng K-D tree không có sự chênh lệch giữa 2 hay 3 chiều

3.2. CÁC THAO TÁC CƠ BẢN VỚI K-D TREE

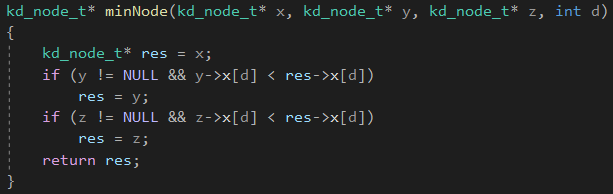
3.2.1. Tìm node nhỏ nhất

Để phục vụ cho việc xóa một node bất kỳ trong K-D tree ta cần phải tìm được node nhỏ nhất. Thuật toán tìm node nhỏ nhất trong 3 node có:

* Input: truyền vào 3 node cần so sánh x, y, z kiểu struct và tọa độ của giá trị cần so sánh là biến d kiểu int
* Output: node nhỏ nhất trong ba node

Hiện thực thuật toán tìm node nhỏ nhất trong 3 node (hàm minNode) gồm các bước:

* Bước 1: Gán giá trị của node x vào node res (gọi node res là node nhỏ nhất)
* Bước 2: Nếu node y có giá trị và giá trị tại tọa độ d của node y nhỏ hơn giá trị tại tọa độ d của node res thì gán node y vào node res.
* Bước 3: Nếu node z có giá trị và giá trị của node z nhỏ hơn giá trị của node res tại cùng tọa độ d thì gán node z vào node res.
* Bước 4: Trả về res.



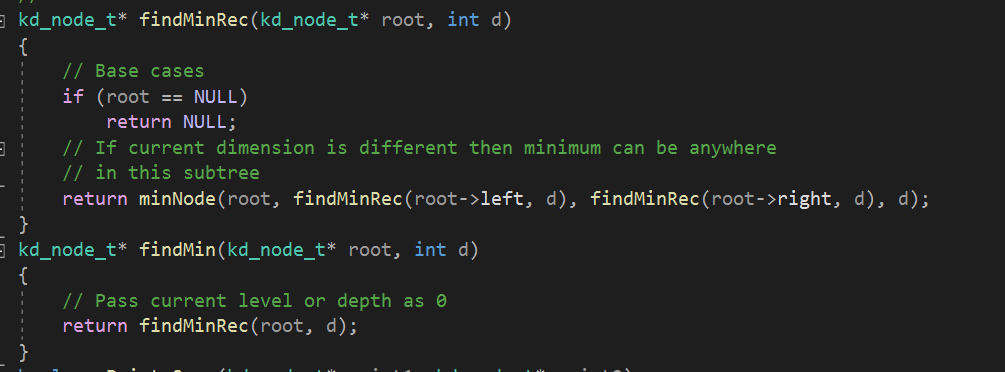
Hình 3.6 Thuật toán tìm node nhỏ nhất trong 3 node

Thuật toán tìm node nhỏ nhất theo nhánh gồm:

* Input: Node hiện tại cần so sánh là node root kiểu struct, tọa độ có giá trị cần so sánh là biến d kiểu int
* Output: Node có giá trị nhỏ nhất trong nhánh

Hiện thực thuật toán tìm node nhỏ nhất theo nhánh (findMind) gồm các bước:

* Bước 1: Nếu node root không có giá trị thì trả về NULL
* Bước 2: Ngược lại thì gọi lại hàm minNode mà em đã đề cập ở trên để trả về node nhỏ nhất trong 3 node: node root, node nhỏ nhất trong nhánh bên trái và node nhỏ nhất trong nhánh bên phải.



Hình 3.7 Thuật toán tìm node nhỏ nhất theo nhánh

3.2.2. Xóa một node ra khỏi K-D tree

Trong thuật toán chính là xóa một node trong K-D tree sẽ bao gồm 2 hàm phụ là hàm arePointSame và CopyPoint:

- Hàm arePointSame dùng để kiểm tra xem 2 node được truyền vào có giống nhau hay không bằng cách chạy vòng lặp for từ i = 0 đến i < 3, nếu xuất hiện tại cùng 1 tọa độ mà giá trị 2 node khác nhau thì sẽ trả về false ngược lại thì sau khi thoát khỏi vòng lặp sẽ trả về true.

- Hàm copyPoint dùng để gán giá trị của node thứ 2 vào node thứ nhất bằng vòng lặp for chạy từ i = 0 đến i < 3, thì trong mỗi lần lặp CT sẽ gán giá trị tại tọa độ i của node thứ hai cho node thứ nhất.

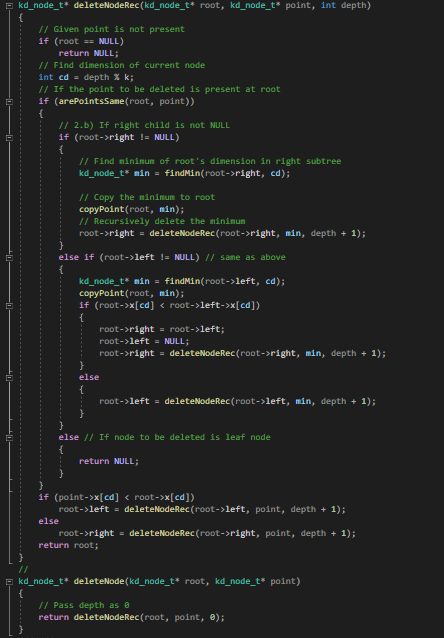
Thuật toán xóa một node ra khỏi K-D tree bao gồm:

- Input: Node hiện tại đang xét là node root, node cần xóa là node point, độ sâu của node root là biến depth

- Output: node gốc của cây sau khi xóa

Hiện thực thuật toán xóa một node ra khỏi K-D tree (deleteNodeRec) gồm các bước:

* Bước 1: Kiểm tra xem node root có giá trị hay không. Nếu không thì trả về NULL
* Bước 2: Xác định tọa độ cd bằng công thức chia depth cho k (k = 3) và lấy phần dư
* Bước 3: Sử dụng hàm arePointsSame để kiểm tra xem node root có bằng node point không. Nếu có thì thực hiện tiếp các bước theo tuần tự, ngược lại nhảy đến bước 7.
* Bước 4: Nếu nhánh bên phải của node root không tồn tại thì thực hiện bước 5 còn ngược lại thì thực hiện các lệnh:
* Thông qua hàm findMin để tìm node nhỏ nhất trong nhánh bên phải của node root và sau đó gán vào node min
* Copy giá trị của node min vào node root bằng hàm copyPoint
* Xóa liên kết của node min vừa tìm được với node root bên nhánh phải
* Bước 5: Nếu nhánh bên trái của node root không tồn tại thì thực hiện bước 6 còn ngược lại thì thực hiện các lệnh:
* Thông qua hàm findMin để tìm node nhỏ nhất trong nhánh bên trái của node root và sau đó gán vào node min
* Copy giá trị của node min vào node root bằng hàm copyPoint
* Nếu giá trị của node root nhỏ hơn giá trị của node bên trái node root thì thực hiện:
  + Tạo liên kết cho node bên trái với nhánh bên phải của node root
  + Xóa liên kết của node bên trái của node root
  + Xóa liên kết của node min vừa tìm được với node root bên nhánh phải
* Ngược lại thì xóa liên kết của node min vừa tìm được với node root bên nhánh trái
* Bước 6: Ngược lại nếu không thực hiện bước 5 thì trả về NULL
* Bước 7: Nếu giá trị của node point cần xóa nhỏ hơn giá trị của node root tại cùng tọa độ cd thì đi tới nhánh bên trái của node root để thực hiện xóa node point bên nhánh trái của node root. Ngược lại thì đi tới nhánh bên phải của node root để xóa node point bên nhánh phải của node root.
* Bước 8: Trả về node root.



Hình 3.8 Thuật toán xóa một node ra khỏi K-D tree

Độ phức tạp về thời gian của thuật toán xóa trong K-D tree:

* Trường hợp tốt nhất và trung bình: O(logN)
* Trường hợp tệ nhất: O(N)

Thời gian chạy của thuật toán xóa trong K-D tree 2 chiều

|  |  |  |  |
| --- | --- | --- | --- |
| N – phần tử | Min | Max | Mean |
| 10 | 0.0000008 | 0.0000028 | 0.00000088 |
| 20 | 0.0000017 | 0.0000033 | 0.0000018 |
| 30 | 0.0000024 | 0.0000045 | 0.0000027 |
| 40 | 0.0000037 | 0.0000075 | 0.0000044 |
| 50 | 0.0000056 | 0.0000109 | 0.0000067 |
| 60 | 0.0000058 | 0.0000229 | 0.0000062 |
| 70 | 0.0000067 | 0.0000288 | 0.0000079 |
| 80 | 0.0000079 | 0.000029 | 0.0000092 |
| 90 | 0.000008 | 0.0000347 | 0.0000116 |
| 100 | 0.0000111 | 0.0000347 | 0.00002 |

Bảng 3.3 Thời gian chạy của thuật toán xóa trong K-D tree 2 chiều

Hình 3.9 Thời gian chạy của thuật toán xóa trong K-D tree 2 chiều

Thời gian chạy của thuật toán tương đối tuân theo hàm tuyến tính

Kết luận: thời gian chạy của thuật xóa tuân theo lý thuyết

So sánh thời gian chạy của thuật xóa một node trong K-D tree 3 chiều:

|  |  |  |
| --- | --- | --- |
| N – phần tử | K-D tree 2 chiều | K-D tree 3 chiều |
| 10 | 0.000001 | 0.000001 |
| 100 | 0.000017 | 0.00002 |
| 1000 | 0.000198 | 0.0002 |
| 100000 | 0.0058 | 0.006 |
| 1000000 | 0.053 | 0.056 |
| 5000000 | 0.0598 | 0.063 |

Bảng 3.4 So sánh thời gian chạy của thuật toán xóa trong K-D tree

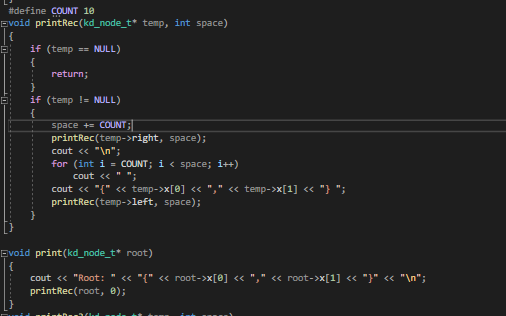
Hình 3.10 So sánh thời gian chạy của thuật toán xóa trong K-D tree

**3.2.3. In K-D tree 2 chiều, 3 chiều**

Từ dữ liệu K-D tree được định nghĩa trong hàm main, in ra màn hình console K-D tree dưới dạng 2D

Thuật toán in K-D tree gồm các bước như sau:

* Bước 1: Đầu tiên kiểm tra K-D tree có giá trị
* Bước 2: Tạo khoảng cách giữa các node trên cây
* Bước 3: Lần lượt in ra các node bên phải, sau đó đến bên trái của cây



Hình 3.11 Thuật toán in K-D tree

Việc in cây 2 chiều, 3 chiều hay nhiều chiều căn bản là giống nhau, chi khác là nếu in cây 2 chiều sẽ có 2 giá trị x được in ra, cây 3 chiều sẽ có 3 giá trị x được in ra

|  |  |
| --- | --- |
| In cây 2 chiều | In cây 3 chiều |
| cout << "{" << temp->x[0] << "," << temp->x[1] << "} "; | cout << "{" << temp->x[0] << "," << temp->x[1] << "," << temp->x[2] << "} "; |

Bảng 3.5 In trong K-D tree 2 chiều và K-D tree 3 chiều

**3.3 ỨNG DỤNG K-D TREE**

### 3.3.1. Ứng dụng K-D tree trong thuật toán tìm kiếm K-Nearest Neighbor

K-nearest neighbor (KNN) là một trong những thuật toán supervised-learning dựa trên các cặp (input, outcome) đã biết từ trước) đơn giản nhất (mà hiệu quả trong một vài trường hợp) trong Machine Learning. Thuật toán dựa trên các điểm giống với nó nhất loại bỏ phần lớn không gian tìm kiếm và nhanh chóng tìm được giá trị.

K-nearest neighbor (KNN) biễu diễn các điểm dữ liệu trong trong không gian k chiều, tính toán khoảng cách quan sát truy vấn và mỗi điểm dữ liệu và tìm điểm K quan sát gần nhất. Các phương pháp tính khoảng cách thường được sử dụng: khoảng cách Euclidean, Mahalanobis và Mahalanobis theo đường chéo.

Trong giới hạn thời gian thực hiện và mục tiêu đề tài, tìm hiểu phương pháp nearest neighbor search trên K-D tree và so sánh với thuật toán Naive K-nearest neighbor để làm rõ ưu, nhược điểm khi sử dụng K-D tree để tìm kiếm nearest neighbor.

**3.3.1.1 Thuật toán tìm kiếm K - Nearest Neighbor trên K-D tree**

Trong nhiều thập kỉ vừa qua dữ liệu thông tin và các thiết bị ngày càng phát triển thúc đẩy việc truy xuất truy xuất thông tin dần được quan tâm. Trong đó thuật toán K Nearest Neighbor - một cải tiến của cây K- D tree là thuật toán phổ biến trong lĩnh vực này.

Thuật toán tìm nearest neighbor này sẽ lợi dụng tính chất của K-D tree để loại bỏ phần lớn các không gian không cần thiết để tìm kiếm nhanh hơn. Và được xây dựng gồm các bước:

Input: node root, điểm cần truy vấn, thứ nguyên của node và chiều của cây

Output: điểm cần tìm, khoảng cách tới điểm gần nhất, các node đi qua.

- Bước 1: Kiểm tra K-D tree từ node root có tồn tại hay không, rồi di chuyển dần xuống các node bên dưới.

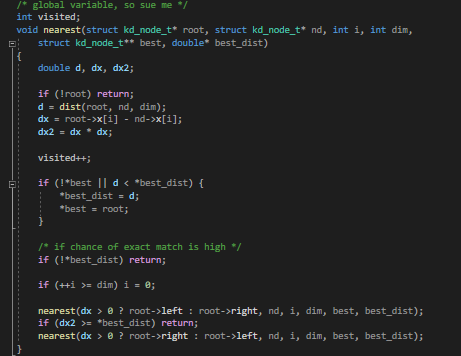
- Bước 2: Gọi d là khoảng cách giữa node root và node đang tìm kiếm gọi là node nd, dx là khoảng cách giữa (root→x[i]) – (nd→x[i]), cho biết node đang tìm kiếm sẽ nhằm bên trái hay bên phải của node root, dx2 là bình phương của dx

- Bước 3: Sử dụng if để kiểm tra node best tồn tại có giá trị hoặc khoảng cách best\_dist có ngắn hơn d không , tiến hành cập nhật thì gán node best = root và khoảng cách gần nhất \*best\_dist = d

- Bước 4: Kiểm tra thứ nguyên i có vượt quá chiều của cây ,nếu cây vượt qua số chiều tiến hành gán lại i=0, tiếp tục đệ quy theo điều kiện dx>0 để xét nhánh bên trái của node root, ngược lại xét nhánh bên phải để tìm điểm gần nhất và khoảng cách gần nhất

- Bước 5: So sánh dx2 ( khoảng cách điểm truy vấn tới bờ) với khoảng cách ngắn nhất đang có, nếu dx2 lớn hơn hoặc bằng thì đã tìm ra khoảng cách ngắn nhất (\*best\_dist ) thì đây chính là điểm cần tìm, ngược lại nếu dx2 nhỏ hơn, dựa vào dx tiếp tục đệ quy để tìm ra khoảng cách ngắn nhất

- Bước 6: mỗi lần đệ quy qua nút của cây biến visited sẽ tăng lên.

****

Hình 3.12 Thuật toán tìm nearest neighbor

Độ phức tạp về thời gian của thuật toán tìm kiếm K - Nearest Neighbor trên K-D tree:

- Trường hợp tốt nhất và trung bình: O(NlogN)

- Trường hợp tệ nhất: O(

|  |  |  |  |
| --- | --- | --- | --- |
| N – phần tử | MIN | MAX | MEAN |
| 10 | 0.0000004 | 0.0000032 | 0.0000005 |
| 100 | 0.0000008 | 0.0000242 | 0.0000011 |
| 1000 | 0.000001 | 0.0001071 | 0.0000024 |
| 10000 | 0.0000015 | 0.00011 | 0.000003 |

Bảng 3.6 Thời gian chạy của thuật toán tìm kiếm K - Nearest Neighbor trên   
K-D tree (2 chiều)

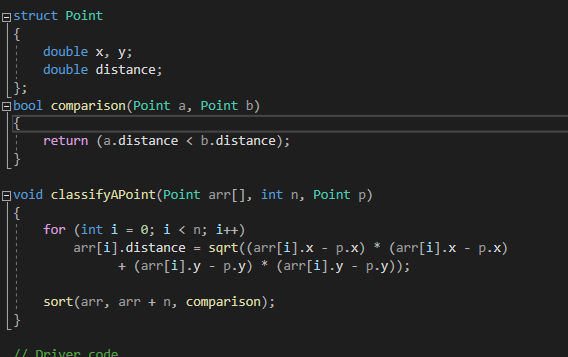
Hình 3.13 Thời gian chạy của thuật toán tìm kiếm K - Nearest Neighbor trên   
K-D tree (2 chiều)

**3.3.1.2 Thuật toán tìm kiếm** **Naive K - Nearest Neighbor**

Là cách đơn giản nhất để tìm điểm gần nhất với điểm N cho trước, được tiến hành bằng cách tính khoảng cách giữa điểm N và tất cả các điểm còn lại. Từ đó, lấy ra được những điểm ứng với khoảng cách K nhỏ nhất. Tốc độ của thuật toán phụ thuộc vào số điểm cho trước. Nếu có N điểm cho trước và M điểm còn lại thì độ phức tạp của thuật toán là O(N\*M), tốc độ chạy thuật toán tăng theo cấp số nhân khi số lượng các điểm tăng lên.

Hiện thực thuật toán Naive K – Nearest Neighbor gồm các bước:

* Bước 1: Đưa dữ liệu một mảng cấu trúc chứa các điểm trong không gian và điểm muốn kiểm tra
* Bước 2: Tính khoảng cách giữa một điểm p muốn kiểm tra với các điểm còn lại
* Bước 3: Sắp xếp các khoảng cách tính được theo thứ tự tăng dần
* Bước 4: Lấy ra các điểm có khoảng cách K nhỏ nhất



Hình 3.14 Thuật toán tìm kiếm Naive K - Nearest Neighbor

|  |  |  |  |
| --- | --- | --- | --- |
| N – phần tử | MIN | MAX | MEAN |
| 10 | 0.0000019 | 0.0000218 | 0.0000028 |
| 100 | 0.0000115 | 0.000197 | 0.000018 |
| 1000 | 0.000105 | 0.000338 | 0.000133 |
| 10000 | 0.00104 | 0.00223 | 0.0016 |

Bảng 3.7 Thời gian chạy của thuật toán Naive K - Nearest Neighbor

Hình 3.15 Thời gian chạy của thuật toán Naive K - Nearest Neighbor

**3.3.1.3 So sánh thuật toán Naive K - Nearest Neighbor và thuật toán K - Nearest Neighbor trên K-D tree**

|  |  |  |
| --- | --- | --- |
| N – phần tử | Naive K – Nearest Neighbor | K - Nearest Neighbor trên K-D tree |
| 0 | 0 | 0 |
| 10 | 0.0000028 | 0.0000005 |
| 100 | 0.000018 | 0.0000011 |
| 1000 | 0.000133 | 0.0000024 |
| 10000 | 0.0016 | 0.000003 |

Bảng 3.8 So sánh thời gian chạy của thuật toán Naive K - Nearest Neighbor và thuật toán K - Nearest Neighbor trên K-D tree (2 chiều)

Hình 3.16 So sánh thời gian chạy của thuật toán Naive K - Nearest Neighbor và thuật toán K - Nearest Neighbor trên K-D tree (2 chiều)

Tiến hành so sánh 2 thuật toán có thể thấy cách tìm cận sử dụng K-D tree sẽ tốn nhiều thời gian để phát triển, cấu trúc nên thuật toán hơn so với cách tiếp cận Naive. Tuy vậy, khi tiến hành so sánh thời gian chạy ta thấy rõ sự tốn kém đó là hoàn toàn xứng đáng.

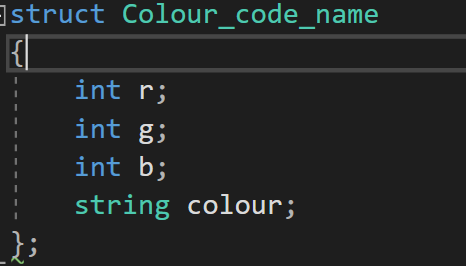
Thời gian của hai thuật toán khá giống nhau đối với các trường hợp nhỏ, nhưng trong các trường hợp lớn như khi N có 1000 phần tử, xuất hiện khác biệt rõ rệt. Trong khi phương pháp tìm K – Nearest Neighbor trên K-D tree có thời gian chạy hầu như không thay đổi. Thì với phương pháp Naive sẽ tốn rất nhiều thời gian để tìm kiếm. Nguyên nhân chủ yếu là do nguyên tắc hoạt động của từng phương pháp, phương pháp Naive phải tiến hành tính khoảng cách giữa điểm muốn kiểm tra với tất cả các điểm còn lại, gây ra sự lãng phí dữ liệu, thời gian. Trong khi phương pháp tìm Nearest Neighbor trên K-D tree sẽ dựa vào các tính năng của K-D tree để phân vùng dữ liệu tìm kiếm, làm giảm thiểu thời gian tính toán, tìm kiếm mang lại ưu điểm rõ rệt khi tìm kiếm dữ liệu lớn.

So sánh hai thuật toán đã cho thấy cách dữ liệu được lưu trữ có thể cực kỳ quan trọng, cùng với ưu điểm của K-D tree khi ứng dụng vào tìm kiếm Nearest Neighbor.

3.3.2. Ứng dụng thuật toán tìm kiếm K-Nearest Neighbor trên K-D tree trong bài toán cụ thể

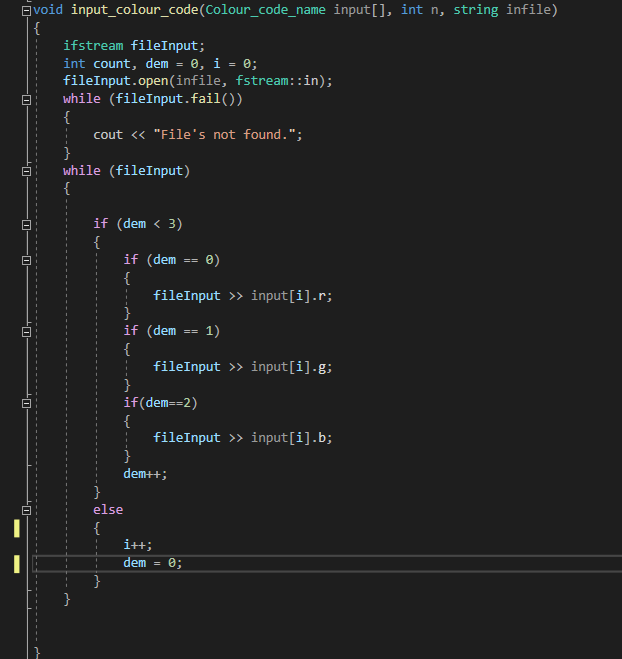
Áp dụng K-D tree 3 chiều để lưu dữ liệu màu dưới dạng RGB từ file, sử dụng thuật toán tìm nearest neighbors search trên K-D tree để tìm ra một điểm màu cận nhất với một mã màu cho trước và in ra mã màu tương ứng với màu tìm được. Các bước thực hiện như sau:

- Bước 1: Tạo kiểu cấu trúc Colour\_code\_name gồm các biến int r, g, b, và string colour.

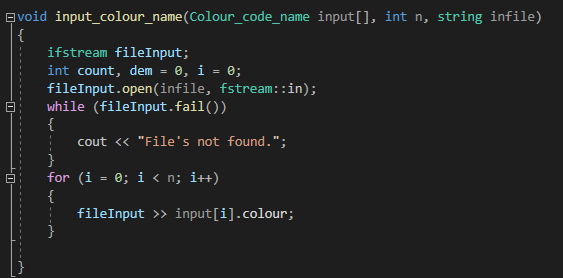


Hình 3.17 Tạo kiểu cấu trúc để in màu

- Bước 2: Dữ liệu gồm màu và tên màu tương ứng theo từng hàng từ 2 file color\_code và color\_name được đưa vào mảng cấu trúc Colour\_code\_name input[].

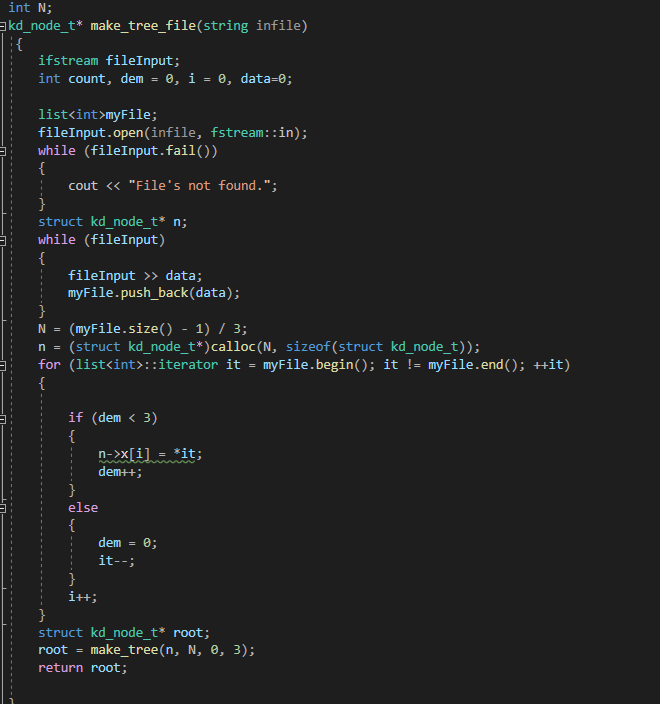


Hình 3.18 Đưa dữ liệu màu vào mảng cấu trúc Colour\_code\_name input[]



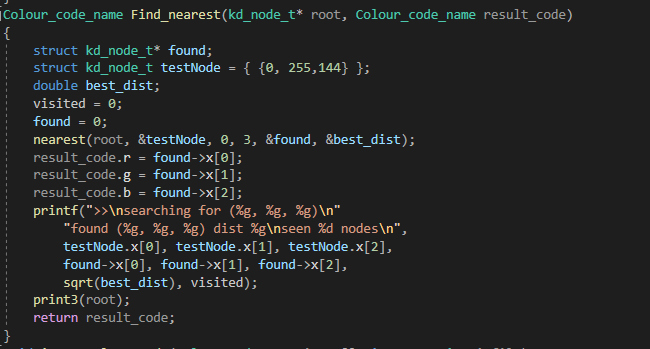
Hình 3.19 Đưa dữ liệu tên màu vào mảng cấu trúc Colour\_code\_name input[]

- Bước 3: Dựng KD-Tree từ file chứa các mã màu tương ứng {x, y, z} bằng hàm kd\_node\_t\* make\_tree\_file.



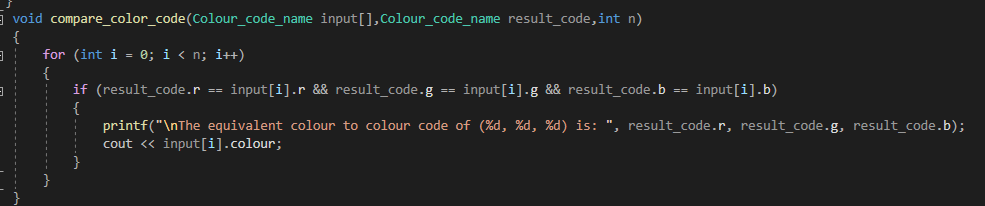
Hình 3.20 Dựng KD-Tree từ file chứa các mã màu

-Bước 4: Áp dụng thuật toán nearest neighbors search để tìm ra mã màu cận nhất với màu đã cho (hàm Colour\_code\_name Find\_nearest) và trả kết quả vào biến Find\_nearest.



Hình 3.21 Áp dụng thuật toán nearest neighbors search để tìm ra mã màu

-Bước 5: So sánh mã màu tìm được (Find\_nearest) với các mã màu trong mảng Colour\_code\_name input[], nếu gặp mã màu trùng thì in ra màu sắc tương ứng và dừng việc tìm kiếm.



Hình 3.22 So sánh mã màu tìm được để tìm ra màu cần tìm kiếm

CHƯƠNG 4: KẾT LUẬN

4.1. KẾ HOẠCH THỰC HIỆN

|  |  |  |
| --- | --- | --- |
| STT | Thời gian | Công việc |
| 1 | 16/10/2021 | Lựa chọn đề tài |
| 2 | 23/10/2021 đến 30/10/2021 | Tìm hiểu, phân tích yêu cầu đề tài và các khái niệm liên quan đến K-D tree |
| 2 | 30/10/2021 đến 6/11/2021 | Dựng code demo K-D tree, tìm hiểu các nội dung cần thực hiện trong báo cáo |
| 3 | 6/11/2021 đến 13/11/2021 | Thêm hàm và chỉnh sửa một số hàm trong code |
| 4 | 13/11/2021 đến 20/11/2021 | Chỉnh sửa một số lỗi trong code, hoàn chỉnh ứng dụng K-D tree |
| 5 | 20/11/2021 đến 27/11/2021 | Thêm các nội dung lý thuyết vào báo cáo, hoàn thiện code K-D tree |
| 6 | 27/11/2021 đến 28/12/2021 | Hoàn thiện nội dung báo cáo, đọc, chỉnh sửa |

Bảng 4.1 Kế hoạch thực hiện đồ án

4.2. KẾT QUẢ ĐẠT ĐƯỢC

- Hiểu được những kiến thức nền tảng về K-D tree, cấu trúc dữ liệu của K-D tree

- Áp dụng được các kiến thức đã tìm hiểu cũng như kiến thức nền từ các môn đã được học trong trường.

- Tìm hiểu được cách xây dựng một K-D tree, cùng các thao tác trên K-D tree, cách xây thuật toán K-nearest neighbor trong K-D tree và xây dựng ứng dụng K-D tree

4.3. KHÓ KHĂN GẶP PHẢI

4.3.1. Công nghệ

|  |  |  |
| --- | --- | --- |
| STT | Khó khăn | Cách khắc phục |
| 1 | Có một số kiến thức mới, lạ chưa được tiếp xúc trước đây | Sử dụng mạng Internet, các trang diễn đàn về công nghệ để tìm hiểu |
| 2 | Quá trình chỉnh sửa, thêm, xóa code thường xảy ra lỗi giữa các các hàm có trước với các hàm được chỉnh sửa, thêm, xóa | Đọc và nghiên cứu kĩ code sẵn có trước khi chỉnh sửa |

Bảng 4.2 Khó khăn về công nghệ

4.3.2. Quá trình thực hiện

|  |  |  |
| --- | --- | --- |
| STT | Khó khăn | Cách khắc phục |
| 1 | Ở những tuần đầu, gặp khó khăn trong việc tìm nguồn tài liệu chất lượng và hướng đi cụ thể. | Cả nhóm cùng giành thời gian bàn bạc và tham khảo ý kiến và tài liệu giảng viên cung cấp |
| 2 | Do dịch bệnh nên không thể gặp mặt trực tiếp và bàn bạc cụ thể về đồ án | Sử dụng các trang mạng xã hội như Facebook, Zalo,...Các nền tảng họp trực tuyến như Google meet |
| 3 | Lịch trình thực hiện đồ án đôi khi bị trễ do lý do khách quan | Tập trung và dành nhiều thời gian hơn cho việc hoàn thành đồ án |

Bảng 4.3 Khó khăn trong quá trình thực hiện

4.4. Kinh nghiệm đạt được

Sau khi hoàn thiện đồ án, các thành viên trong nhóm rút ra nhiều bài học để thực hiện các dự án sau tốt hơn:

* Chú ý tới việc quản lý thời gian khi làm đồ án, thường xuyên liên lạc, nhắc nhở nhau hoàn thiện công việc
* Lưu trữ, sao chép code ở nhiều nơi để khi xảy ra sự cố không cần phải xây dựng lại từ đầu
* Cần chủ động xóa các tính năng không cần thiết trong code, đọc hiểu các phương thức của code trước khi thêm các tính năng để tránh lỗi không cần thiết

TÀI LIỆU THAM KHẢO

* + - 1. Computer science(2014) truy cập ngày 3/12/2021 tại: <http://www.cs.siue.edu/~marmcke/docs/cs490/spatialIndexing.html>
      2. Javatpoint, “K-Nearest Neighbor(KNN) Algorithm for Machine Learning”, truy cập ngày 4/12/2021 tại: <https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning>
      3. Jerome H. Friedman -[Raphael Finkel](https://www.researchgate.net/profile/Raphael-Finkel) - [Jon Louis Bentley](https://www.researchgate.net/scientific-contributions/Jon-Louis-Bentley-4782728) (1977), “An Algorithm for Finding Best Matches in Logarithmic Expected Time”, truy cập ngày 26/10/2021 tại: [(PDF) An Algorithm for Finding Best Matches in Logarithmic Expected Time (researchgate.net)](https://www.researchgate.net/publication/220493118_An_Algorithm_for_Finding_Best_Matches_in_Logarithmic_Expected_Time)
      4. Kevin MacDonald (2020), “Comparison of Naive KNN and K-D Tree KNN”, truy cập ngày 21/12/2021 tại: <https://kevinmacdonald.me/post/kdtree_comparison/>
      5. Yasen Hu (2019), “KD-Tree”, truy cập ngày 4/12/2021 tại: <https://yasenh.github.io/post/kd-tree/?fbclid=IwAR1QhidXy6_oQXrxhh71RasujDZD3zcvwd1ZjK59_mhBKAum9WFnxN1n0NU>
      6. Nguồn code tham khảo: <https://rosettacode.org/wiki/K-d_tree#C.2B.2B>

# **LINK GITHUB CỦA ĐỒ ÁN**

1. https://github.com/nhudang1511/Report-8-K-D-tree\_team-2