ỨNG DỤNG CÁC KĨ THUẬT CỦA HÌNH HỌC TÍNH TOÁN ĐỂ XÂY DỰNG CÂY PHẠM VI HỖ TRỢ BÀI TOÁN TRUY VẤN CƠ SỞ DỮ LIỆU

Lê Thị Thuấn*, Phạm Minh Tuyến
Trường Đại học Khánh Hòa

Thông tin chung:
Ngày nhận bài: 02/09/2024
Ngày phản biện: 05/09/2024
Ngày duyệt đăng: 22/09/2024
**Tác giả liên hệ:*
lethithuan@ukh.edu.vn

Title:
Applying computational geometry techniques to build data structure for range trees to assist the query for database.

Từ khóa:
Hình học tính toán, Phạm vi truy vấn, cây phạm vi, tập điểm chung

Keywords:
Computational geometry, Range searching, Range trees, General sets of points

TÓM TẮT: *Cuộc cách mạng công nghiệp 4.0 đã mở ra một thời đại cho công nghệ số, vì vậy hệ thống dữ liệu được lưu trữ trên các hệ quản trị cơ dữ liệu càng lớn và phức tạp. Mối tương quan giữa tính chất hình học và cơ sở dữ liệu là phương pháp chuyển đổi các bản ghi trong một cơ sở dữ liệu thành các điểm trong không gian đa chiều và chuyển đổi các truy vấn về các bản ghi thành các truy vấn lên tập các điểm này. Trong bài báo này, chúng tôi tập trung đề xuất xây dựng cấu trúc cây phạm vi bằng các kỹ thuật của hình học tính toán để thực hiện truy vấn trên phạm vi trực giao, đồng thời cũng đề xuất giải pháp cải tiến thuật toán cây phạm vi bằng phương pháp tập điểm chung. Để chứng minh cho cơ sở lý đã đề xuất, chúng tôi tiến hành xây dựng chương trình thử nghiệm tính hiệu quả của thuật toán trên các tập dữ liệu khác nhau. Kết quả thực nghiệm được so sánh với cây KD trên cùng một bộ dữ liệu nhằm minh chứng phương pháp đề xuất mới của chúng tôi được tối ưu hóa hơn về thời gian tìm kiếm.*

ABSTRACT: *The Fourth Industrial Revolution is the trend towards digital technologies to lead large databases and complicated database storages on Database Management System. At first sight it seems that databases have little to do with geometry. Nevertheless, queries about data in a database can be interpreted geometrically. To this end we transform records in a database into points in a multi-dimensional space, and we transform the queries about the records into queries on this set of points. This paper proposed an approach based on computational geometry techniques for building range trees, two-dimensional range search algorithm (search range trees algorithm) and general sets of points to improve the algorithm. To prove for proposed theorem, we had experimental research of Range-Tree structure on the different datasets. The experimental results of an orthogonal range query evaluated with the recently published method of KD-Tree on the same dataset. This proof shows that our proposed method is more time-efficient in queries.*

1. Giới thiệu vấn đề nghiên cứu

Hình học tính toán nổi lên từ lĩnh vực phân tích và thiết kế thuật toán trong cuối những năm 1970 và phát triển thành một bộ môn khoa học. Sự thành công của hình học tính toán không chỉ trong nghiên cứu và đưa ra được các giải pháp để giải quyết các bài toán hình học, mà còn được ứng dụng trong nhiều lĩnh vực như đồ họa máy tính, các hệ thống thông tin địa lý, người máy và các lĩnh vực khác [1] [2]. Để giải quyết các bài toán có tính chất hình học chủ yếu dựa trên hai thành phần. Một là sự hiểu biết thấu đáo các tính chất hình học của bài toán, hai là ứng dụng thích hợp các kỹ thuật thuật toán và cấu trúc dữ liệu. Các bước để xây dựng một thuật toán hình học bao gồm: (i) Phân tích và thiết kế thuật toán sơ bộ ban đầu; (ii) Hiệu chỉnh sao cho thuật toán đúng đắn khi xuất hiện các trường hợp suy biến; (iii) Thực thi thuật toán

[3].

Đối với lĩnh vực lưu trữ dữ liệu, ngày càng nhiều kho dữ liệu và các hệ thống phân tích trực tuyến được sử dụng trong nhiều công ty để lấy ra và phân tích những thông tin có lợi từ các cơ sở dữ liệu rất lớn nhằm đưa ra các quyết định [4]. Các kỹ thuật cơ sở dữ liệu động và thời gian thực được sử dụng trong việc kiểm tra các tiến trình công nghiệp và sản xuất [5]. Vì vậy các hệ cơ sở dữ liệu cũng theo đó tăng lên cả về dung lượng lẫn tính phức tạp của dữ liệu. Lúc này nảy sinh vấn đề là làm thế nào để việc tìm kiếm trên các cơ sở dữ liệu, đặc biệt là trên các cơ sở dữ liệu lớn, truy vấn một cách nhanh chóng, chính xác và tốn ít tài nguyên? Đây là bài toán luôn được các nhà khoa học cũng như các nhà ứng dụng quan tâm [6][7]. Xét một cơ sở dữ liệu phục vụ cho quản lý nhân sự, lưu trữ tên, ngày sinh, thu nhập ... của mỗi nhân viên. Ví dụ về một dạng truy vấn thông thường lên cơ sở dữ liệu nói trên là liệt kê tất cả các nhân viên sinh từ năm 1980 đến năm 1985, có thu nhập hàng tháng từ 3000 đến 4000 đô la và có từ 2 đến 4 đứa con. Trong trường hợp này, xem mỗi nhân viên là một điểm trong không gian ba chiều: tọa độ đầu tiên đại diện cho ngày sinh, tọa độ thứ hai là thu nhập, và tọa độ thứ ba là số con. Kết quả của truy vấn chính là đi liệt kê tất cả các điểm nằm trong hình hộp chữ nhật $[19500000: 19559999] \times [3000: 4000] \times [2: 4]$, một truy vấn kiểu như thế này được gọi là *truy vấn phạm vi trực giao* trong hình học tính toán [8][9]. Thực hiện truy vấn với phạm vi hai chiều, khi thực hiện trên cây KD với thời gian tìm kiếm $O(\sqrt{n} + k)$ [10]. Trong cách tiếp cận mới của chúng tôi, sử dụng cấu trúc cây phạm vi để thực hiện tìm kiếm trên phạm vi trực giao hai chiều với thời gian truy vấn tốt hơn $O(\log n + k)$, đồng thời đề xuất phương pháp tập điểm chung để cải tiến thuật toán trong trường hợp có một số nhân viên có những thông tin trùng nhau. Thuật toán được trình bày chi tiết bởi các bước và được minh họa bằng tập các điểm trong không gian hai chiều.

2. Cơ sở lý thuyết và phương pháp nghiên cứu

2.1. Phương pháp tìm kiếm trên cây phạm vi

Cho P là tập n điểm trong mặt phẳng và phạm vi tìm kiếm $[x: x'] \times [y: y']$. Yêu cầu liệt kê các điểm từ P mà nằm trong phạm vi trực giao $[x: x'] \times [y: y']$.

Thực hiện truy vấn phạm vi trực giao $[x: x'] \times [y: y']$ chính là tổ hợp của hai truy vấn con một chiều: một theo tọa độ x và một theo tọa độ y của các điểm. Điều này đưa đến ý tưởng phân tách tập điểm đã cho lần lượt theo tọa độ x và tọa độ y. Nghiên cứu tiến hành tìm kiếm trên phạm vi một chiều, là cơ sở để xây dựng cây phạm vi và thực hiện tìm kiếm trên phạm vi trực giao.

2.1.1. Tìm kiếm trên phạm vi một chiều

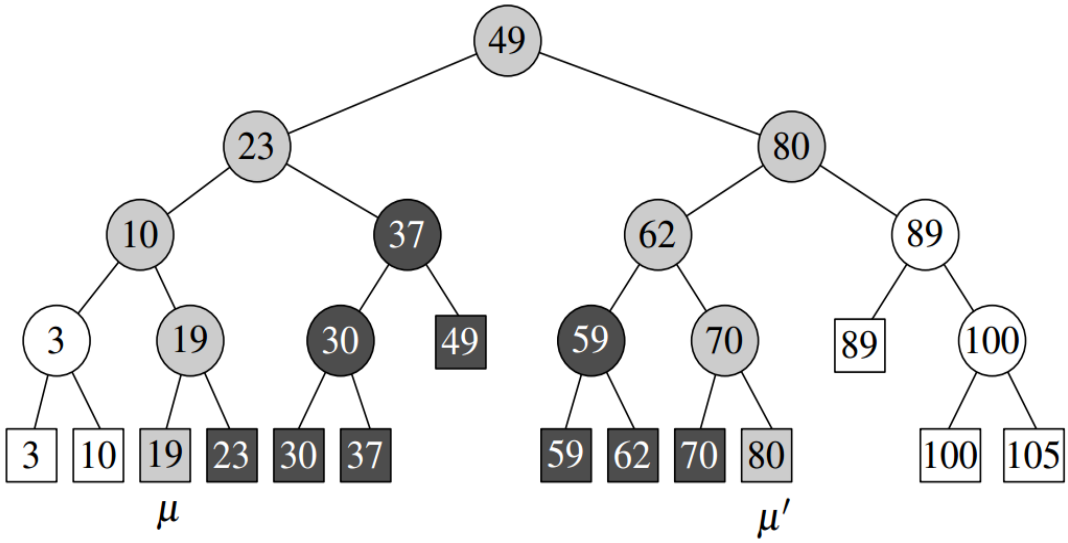
Cho một tập hợp số thực là các điểm trong không gian một chiều. Yêu cầu của một truy vấn đưa ra là liệt kê các điểm nằm bên trong một phạm vi truy vấn trực giao một chiều, tức là nằm bên trong một đoạn $[x: x']$.

Gọi $P := \{p_1, p_2, \dots, p_n\}$ là tập các điểm trên đường thẳng thực đã cho. Phương pháp giải quyết bài toán là sử dụng cấu trúc dữ liệu cây nhị phân tìm kiếm cân bằng T[11]. Các lá của T lưu trữ các điểm của P và các nút trong của T lưu trữ các giá trị phân tách để chỉ dẫn tìm kiếm. Ký hiệu giá trị phân tách được chứa tại một nút υ là x_υ. Giả thiết rằng cây con bên trái của một nút υ chứa tất cả các điểm nhỏ hơn hoặc bằng x_υ, và cây con bên phải chứa tất cả các điểm lớn hơn x_υ.

Để liệt kê các điểm trong một phạm vi trực giao $[x: x']$, nghiên cứu thực hiện như sau: Tìm kiếm x và x' trong T. Giả sử μ và μ′

tương ứng là hai lá nơi mà việc tìm kiếm kết thúc. Khi đó các điểm nằm trong khoảng $[x: x']$ là những điểm được chứa trong những lá nằm giữa μ và μ′, có thể bao gồm cả điểm được chứa tại μ và điểm được chứa tại μ′.

Ví dụ:



Hình 1. Tìm kiếm phạm vi một chiều trong cây nhị phân tìm kiếm

Thực hiện tìm kiếm với khoảng [18: 77] trong cây ở hình 1, tức là phải liệt kê tất cả các điểm được chứa trong những lá màu xám [12][13].

Làm thế nào để có thể tìm những lá nằm giữa μ và μ′? Như hình 1 đã gợi ý, đó là những lá của các cây con nằm giữa các đường dẫn tìm kiếm tới μ và μ′. Cụ thể, những cây con này có màu xám đậm, còn các nút nằm trên các đường dẫn tìm kiếm có màu xám nhạt. Để tìm những nút này trước tiên phải tìm nút $\upsilon_{\text{phân tách}}$ mà tại đó các đường dẫn tới x và x' phân tách. Thủ tục con sau đây sẽ thực hiện điều này, với giả thiết $lc(\upsilon)$ và $rc(\upsilon)$ tương ứng là nút con bên trái và bên phải của nút υ.

Thủ tục: Tim_nut_phan_tach(T, x, x')

Đầu vào: Một cây T và hai giá trị x, x', với $x \leq x'$

Đầu ra: Nút υ mà tại đó các đường dẫn tới x và x' phân tách, hoặc là lá mà tại đó cả hai đường dẫn tìm kiếm kết thúc.

Các bước:

1. $\upsilon \leftarrow \text{root}(T)$

2. WHILE υ không phải là một lá **AND** $(x' \leq x_\upsilon$ **OR** $x > x_\upsilon)$

3. **DOIF** $x' \leq x_\upsilon$

4. **THEN** $\upsilon \leftarrow lc(\upsilon)$

5. **ELSE** $\upsilon \leftarrow rc(\upsilon)$

6. RETURN υ

Hình 2. Thuật toán tìm nút phân tách

Bắt đầu từ $\upsilon_{\text{phân tách}}$ đi theo đường dẫn tìm kiếm x. Tại mỗi nút mà tại đó đường dẫn đi về bên trái thì liệt kê tất cả các lá nằm trong cây con bên phải, bởi vì cây con này là cây con nằm giữa hai đường dẫn tìm kiếm. Tương tự, đi theo đường dẫn tìm kiếm và liệt kê các lá nằm trong cây con bên trái của các nút mà tại đó đường dẫn đi về bên phải. Cuối cùng, kiểm tra các điểm được chứa tại các lá mà tại đó các đường dẫn kết thúc, chúng có thể thuộc hoặc không thuộc phạm vi $[x: x']$

Trong thuật toán sử dụng một thủ tục con Liet_ke_cay_con để duyệt qua cây con có gốc tại một nút cho trước và liệt kê các điểm được chứa tại các lá của nó.

Thủ tục: Liet_ke_cay_con(υ)

Đầu vào: Một nút υ của cây nhị phân tìm kiếm T.

Đầu ra: Danh sách các điểm được chứa tại các lá của cây con có gốc là υ.

Các bước:

1. IFυ là một lá

2. **THEN**Liệt kê điểm được chứa tại υ.

3. **ELSE IF**υ không rỗng

4. **THEN** Liet_ke_cay_con(lc(υ))

5. Liet_ke_cay_con(rc(υ))

6. RETURNυ

Hình 3. Thuật toán liệt kê cây con

Sau khi liệt kê được các nút lá thì thuật toán tìm kiếm trên phạm vi một chiều thực hiện như sau:

Thuật toán:

TimkiemPhamvi1Chieu(T, $[x: x']$)

Đầu vào: Một cây nhị phân tìm kiếm T và một phạm vi tìm kiếm $[x: x']$

Đầu ra: Tất cả các điểm được chứa trong T và nằm trong phạm vi tìm kiếm.

Các bước:

$\upsilon_{\text{phân tách}} \leftarrow$ Tim_nut_phan_tach(T, x, x')

1. IF$\upsilon_{\text{phân tách}}$là một lá

2. **THEN**Kiểm tra nếu điểm được chứa tại $\upsilon_{\text{phân tách}}$ là điểm cần phải được liệt kê thì liệt kê nó.

3. **ELSE** (*Đi theo đường dẫn tới x và liệt kê các điểm trong các cây con nằm bên phải đường dẫn*)

4. $\upsilon \leftarrow lc(\upsilon_{\text{phân tách}})$

5. **WHILE**υ không phải là một lá

6. **DO IF**$x \leq x_\upsilon$

7. **THEN**Liet_ke_cay_con($rc(\upsilon)$)

8. $\upsilon \leftarrow lc(\upsilon)$

9. **ELSE** $\upsilon \leftarrow rc(\upsilon)$

10. Kiểm tra nếu điểm được chứa tại lá υ là điểm cần phải được liệt kê thì liệt kê nó.

11. Tương tự, đi theo đường dẫn tới x', liệt kê các điểm trong các cây con nằm bên trái đường dẫn và kiểm tra điểm được chứa tại lá nơi mà đường dẫn kết thúc có phải là điểm cần phải được liệt kê hay không, nếu phải thì liệt kê nó.

Hình 4. Thuật toán liệt kê cây con

2.1.2. Tìm kiếm trên phạm vi trực giao hai chiều - cây phạm vi

Cho P là tập n điểm trong mặt phẳng và phạm vi tìm kiếm $[x: x'] \times [y: y']$.

Đầu tiên tập trung tìm kiếm những điểm mà tọa độ x của nó nằm trong phạm vi $[x: x']$ – tức là khoảng x của vùng chữ nhật tìm kiếm, còn đối với tọa độ y sẽ tính toán sau. [13][14]

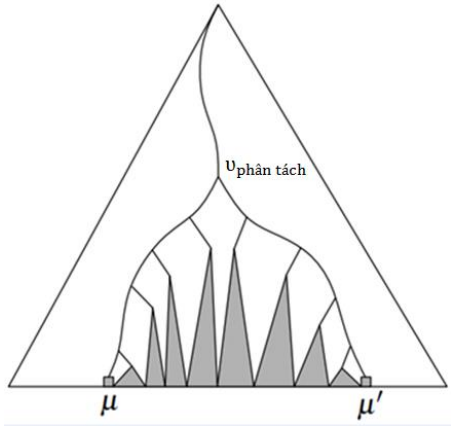
Nếu chỉ quan tâm đến tọa độ x thì yêu cầu tìm kiếm trở thành một yêu cầu tìm kiếm phạm vi một chiều và sử dụng một cây nhị phân tìm kiếm trên tọa độ x của các điểm. Nói một cách khái quát thì thuật toán tìm kiếm như sau:

Tìm kiếm với x và x' trong cây, cho đến khi đưa ra được một nút $\upsilon_{\text{phân tách}}$ mà tại đó đường dẫn tìm kiếm bị phân tách.

Từ nút con bên trái của $\upsilon_{\text{phân tách}}$ tiếp tục tìm kiếm với x, và tại mỗi nút υ nơi mà đường dẫn tìm kiếm x đi về bên trái, liệt kê tất cả các điểm trong cây con bên phải của υ.

Tương tự, tiếp tục tìm kiếm với x' từ nút con bên phải của $\upsilon_{\text{phân tách}}$, và tại mỗi nút υ nơi mà đường dẫn tìm kiếm x' đi về bên phải, liệt kê tất cả các điểm trong cây con bên trái của υ.

Cuối cùng, kiểm tra những lá μ và μ' – nơi mà hai đường dẫn kết thúc – để xem chúng có chứa điểm trong phạm vi tìm kiếm hay không. Để có thể hình dung những gì mô tả ở trên, hãy hình dưới đây (hình 5):



Hình 5. Tìm kiếm một chiều theo tọa độ x

Kết quả của thuật toán trên là lựa chọn được một tập $O(\log n)$ cây con mà đều chứa chính xác các điểm có tọa độ x nằm trong khoảng x của vùng chữ nhật tìm kiếm.

Gọi tập con các điểm được lưu trữ trong các lá của cây con có gốc tại υ là *tập con chính tắc* [12][13] của υ. Ví dụ, tập con chính tắc của gốc cây là toàn bộ tập P, còn tập con chính tắc của một lá là điểm được lưu trữ tại lá đó.

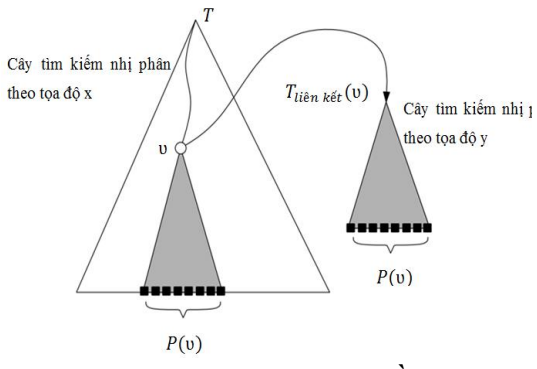
Ký hiệu tập con chính tắc của nút υ là $P(\upsilon)$.

Nhận thấy, tập con các điểm mà tọa độ x của chúng thuộc phạm vi tìm kiếm có thể được biểu diễn như là *hợp rời* của $O(\log n)$ tập con chính tắc, đây là các tập $P(\upsilon)$ của các nút υ mà đóng vai trò là gốc của các cây con được lựa chọn.

Nghiên cứu tìm hết tất cả các điểm nằm trong những tập con chính tắc của $P(\upsilon)$ mà chỉ muốn liệt kê những điểm nào có tọa độ y thuộc phạm vi $[y: y']$. Đây là một yêu cầu tìm kiếm một chiều khác, nên chúng tôi sẽ sử dụng cây nhị phân tìm kiếm trên tọa độ y của các điểm trong $P(\upsilon)$. Điều này đưa đến cấu trúc dữ liệu cho các truy vấn phạm vi trực giao lên một tập P của n điểm trong mặt phẳng như sau:

- Cây chính là một cây nhị phân tìm kiếm cân bằng T xây dựng trên tọa độ x của các điểm trong P.
- Với mọi nút trong hay nút lá của T, tập con chính tắc $P(\upsilon)$ được lưu trữ trong một cây nhị phân tìm kiếm cân bằng $T_{\text{liên kết}}(\upsilon)$ trên tọa độ y của các điểm. Nút υ lưu trữ một con trỏ chỉ tới gốc của $T_{\text{liên kết}}(\upsilon)$. Cây $T_{\text{liên kết}}(\upsilon)$ này được gọi là cấu trúc liên kết của υ.

Cấu trúc dữ liệu trên được gọi là một cây phạm vi. Hình 6 minh họa cấu trúc của một cây phạm vi.



Hình 6. Một cây phạm vi hai chiều

Những cấu trúc dữ liệu mà các nút chứa con trỏ chỉ tới các cấu trúc liên kết thường được gọi là những cấu trúc dữ liệu đa tầng. Cây chính T được gọi là cây tầng 1, còn các cấu trúc liên kết được gọi là các cây tầng 2.

Một cây phạm vi có thể được xây dựng với thuật toán đệ qui sau:

Lưu ý: Trong thuật toán này, đầu vào được xem như tập các điểm được sắp xếp theo tọa độ x và đầu ra là gốc của cây phạm vi 2 chiều T của P. Chúng tôi giả định là không có 2 điểm nào có cùng tọa độ x hay y (giả định này sẽ khắc phục bằng phương pháp tập điểm chung trong mục 2.2).

Thuật toán Xay_dung_cay_pham_vi_2D(P)

Đầu vào: Một tập P của các điểm trong mặt phẳng

Đầu ra: Gốc của một cây phạm vi hai chiều.

Các bước:

1. Xây dựng cấu trúc liên kết: Xây dựng một cây nhị phân tìm kiếm $T_{\text{liên kết}}$ trên tập P_y (tập tọa độ y của các điểm trong P). Lưu trữ tại các lá của $T_{\text{liên kết}}$ không chỉ tọa độ y của các điểm trong P_y mà còn cả chính các điểm này.
2. **If** P chỉ chứa một điểm
3. **then** Tạo một lá υ lưu trữ điểm này và gán $T_{\text{liên kết}}$ làm cấu trúc liên kết của υ.
4. **else** Phân tách P thành hai tập con; một tập con $P_{\text{trái}}$ chứa các điểm có tọa độ x nhỏ hơn hoặc bằng x_{tb}, và tập con $P_{\text{phải}}$ chứa các điểm có tọa độ x lớn hơn x_{tb}, với x_{tb} là trung bình tọa độ x.
5. $\upsilon_{\text{trái}} \leftarrow$ y_dung_cay_pham_vi_2D($P_{\text{trái}}$)
6. $\upsilon_{\text{phải}} \leftarrow$ Xay_dung_cay_pham_vi_2D($P_{\text{phải}}$)
7. Tạo một nút υ chứa x_{tb}, gán $\upsilon_{\text{trái}}$ làm con trái của υ, $\upsilon_{\text{phải}}$ làm con phải của υ, $T_{\text{liên kết}}$ làm cấu trúc liên kết của υ.
8. **Return** υ

Hình 7. Thuật toán xây dựng cây phạm vi

Lưu ý là trong các lá của các cấu trúc liên kết không chỉ lưu trữ tọa độ y của các điểm mà còn cả chính các điểm đó. Điều này là cần thiết, bởi vì khi tìm kiếm các cấu trúc liên kết, cần phải liệt kê các điểm chứ không chỉ là liệt kệ các tọa độ y.

Tiếp theo, chúng tôi xây dựng thuật toán tìm kiếm cho cây phạm vi hai chiều, thuật toán lựa chọn $O(\log n)$ tập con chính tắc mà đều chứa các điểm có tọa độ x thuộc phạm vi $[x: x']$. Có thể thực hiện điều này với thuật

toán tìm kiếm một chiều. Sau đó, với những tập con chính tắc này sẽ liệt kê các điểm mà có tọa độ y thuộc phạm vi $[y: y']$. Để thực hiện bước này chúng tôi cũng sử dụng thuật toán tìm kiếm một chiều, được áp dụng cho các cấu trúc liên kết mà lưu trữ các tập con chính tắc đã chọn. Có thể thấy rằng thuật toán tìm kiếm lúc này hầu như tương tự với thuật toán tìm kiếm một chiều TimkiemPhamvi1Chieu, chỉ khác ở chỗ thay vì gọi tới thủ tục Liet_ke_cay_con thì nó sẽ gọi tới TimkiemPhamvi1Chieu.

Thuật toán

TimkiemPhamvi2Chieu_CayPhamvi($T, [x: x'] \times [y: y']$)

Đầu vào: Một cây phạm vi 2 chiều T và một phạm vi tìm kiếm $[x: x'] \times [y: y']$.

Đầu ra: Tất cả các điểm trong T và thuộc phạm vi tìm kiếm.

Các bước:

1. $\upsilon_{\text{phân tách}} \leftarrow$Tim_nut_phan_tach($T, x, x'$)
2. **if** $\upsilon_{\text{phân tách}}$ là một lá
3. **then** Kiểm tra nếu điểm lưu trữ tại $\upsilon_{\text{phântách}}$ thuộc phạm vi tìm kiếm thì liệt kê nó.
4. **else** (* Đi theo đường dẫn tới x và gọi TimkiemPhamvi1Chieu trên các cây con bên phải của đường dẫn. *)
5. $\upsilon \leftarrow lc(\upsilon_{\text{phân tách}})$
6. **while**υkhông phải là một lá
7. **do if** $x \leq x_{\upsilon}$
8. **then**TimkiemPhamvi1Chieu($T_{\text{liên kết}}(rc(\upsilon)), [y: y']$)
9. $\upsilon \leftarrow lc(\upsilon)$
10. **else** $\upsilon \leftarrow rc(\upsilon)$
11. Kiểm tra nếu điểm lưu trữ tại υthuộc phạm vi tìm kiếm thì liệt kê nó.
12. Tương tự, đi theo đường dẫn từ $rc(\upsilon_{\text{phân tách}})$ tới x', gọi TimkiemPhamvi1Chieu với phạm vi $[y: y']$ trên các cấu trúc liên kết của các cây con bên phải của đường dẫn và kiểm tra nếu điểm lưu trữ tại lá nơi đường dẫn kết thúc thuộc phạm vi tìm kiếm thì liệt kê nó.

Hình 8. Thuật toán tìm kiếm trên cây phạm vi

Để xây dựng cây phạm vi và tìm kiếm trên cây phạm vi từ tập P gồm n điểm trong mặt phẳng thì cần thời gian lưu trữ là $O(nlogn)$ và thời gian tìm kiếm để liệt kê các điểm thuộc phạm vi trực giao là $O(log^2 n + k)$ với k là số điểm được liệt kê.

2.2. Giải pháp cải tiến

Trong phần trước, khi tìm kiếm trên cây phạm vi giả thiết rằng không có bất kỳ hai điểm nào có cùng tọa độ *x* hay *y*, đây là một giả thiết phi thực tế. Ví dụ, khi các điểm lưu trữ thông tin về mức thu nhập và số con của nhân viên thì không thể tránh khỏi việc có một số nhân viên có những thông tin trùng nhau. Để giải quyết vấn đề này, nghiên cứu đề xuất giải pháp tập điểm chung để cải tiến thuật toán tìm *kiếm trên cây phạm vi*, được thực hiện như sau: thay thế các tọa độ số thực bởi các phần tử của không gian hợp số. Các phần tử của không gian này là các cặp số thực. Hợp số của hai số thực a và b được ký hiệu là (a/b). Định nghĩa thứ tự trong không gian hợp số như sau: với hai hợp số (a/b) và (a'/b'):

$(a/b) < (a'/b') \Leftrightarrow a < a'$ hoặc ($a = a'$ và $b < b'$).

Giả thiết *P* là tập *n* điểm đã cho trong mặt phẳng. Các điểm riêng biệt, nhưng nhiều điểm có thể có cùng tọa độ *x* hoặc *y*. Thay thế mỗi điểm $p := (p_x, p_y)$ bởi một điểm mới $\hat{p} := ((p_x|p_y), (p_y|p_x))$ sao cho có các hợp số như là các giá trị tọa độ. Bây giờ dựa vào tập $\hat{P}$ hãy xây dựng cây phạm vi.

Giả sử như muốn liệt kê các điểm của *P* mà nằm trong một phạm vi $R := [x: x'] \times [y: y']$. Để thực hiện điều này thì phải tìm kiếm trên cây đã xây dựng cho $\hat{P}$. Có nghĩa rằng phải thực hiện việc biến đổi phạm vi tìm kiếm sang không gian hợp số mới. Phạm vi biến đổi $\hat{R}$ được định nghĩa như sau:

$\hat{R} := [(x|-\infty): (x'|+\infty)] \times [(y|-\infty): (y'|+\infty)]$.

3. Kết quả nghiên cứu và bàn luận

Công cụ lập trình được lựa chọn trong nghiên cứu này là Microsoft Visual Studio 2015.

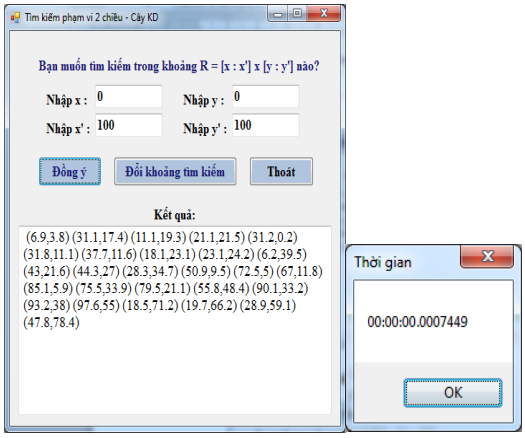
Đây là chương trình thực nghiệm tính hiệu quả của thuật toán tìm kiếm sử dụng cấu trúc dữ liệu cây phạm vi và cây KD khi áp dụng phương pháp sử dụng tập điểm chung để khắc phục tình huống các điểm trong tập điểm đầu

vào có cùng tọa độ x hoặc tọa độ y.

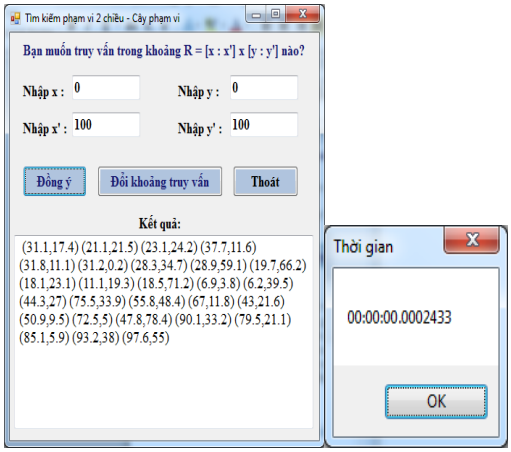
Dữ liệu vào là một tập P chứa các điểm trong mặt phẳng. Tọa độ x hay tọa độ y của mỗi điểm trong tập P là một số thực ngẫu nhiên trong phạm vi từ 0 tới 1000.

Nhiệm vụ của chương trình là liệt kê tất cả các điểm thuộc tập P mà nằm trong phạm vi tìm kiếm $[x: x'] \times [y: y']$ đã áp dụng phương pháp tập điểm chung để khắc phục tình huống các điểm trong tập điểm đầu vào có cùng tọa độ x hoặc tọa độ y

Thử nghiệm1: với phạm vi tìm kiếm $[0: 100] \times [0: 100]$ chúng tôi thu được kết quả như sau:



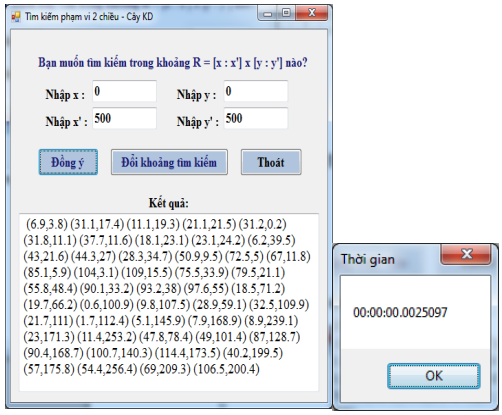
Hình 9. Kết quả 1 - tìm kiếm trên Cây KD



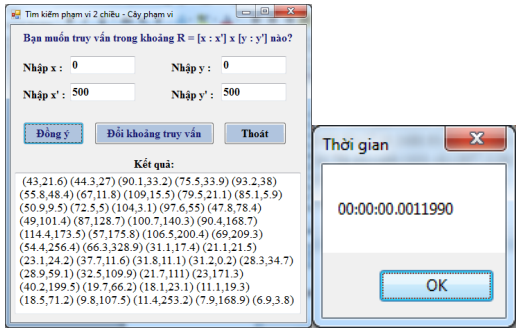
Hình 10. Kết quả 1 - tìm kiếm trên Cây phạm vi

Dựa trên kết quả thực nghiệm trên có thể đưa ra nhận xét rằng cả hai thuật toán đều cho kết quả trong phạm vi tìm kiếm $[0: 100] \times [0: 100]$, nhưng tốc độ tìm kiếm của thuật toán sử dụng cấu trúc dữ liệu Cây phạm vi nhanh hơn. Để kiểm tra lại khẳng định này, chúng tôi đổi các khoảng tìm kiếm khác:

Kết quả tìm kiếm với phạm vi tìm kiếm $[0: 500] \times [0: 500]$:



Hình 11. Kết quả 2 - tìm kiếm trên Cây KD



Hình 12. Kết quả 2 - tìm kiếm trên Cây phạm vi

Như vậy, xét về độ chính xác thì cả hai thuật toán tìm kiếm trên phạm vi hai chiều sử dụng cấu trúc dữ liệu *Cây KD* và *Cây phạm vi* là tương đương nhau. Nhưng, như đã đánh giá khi nghiên cứu lý thuyết, thời gian tìm kiếm của thuật toán sử dụng cấu trúc *Cây phạm vi* ngắn hơn, và tốc độ tìm kiếm của cả hai thuật toán tỉ lệ thuận với số điểm được liệt kê.

4. Kết luận

Thuật toán tìm kiếm phạm vi hai chiều – cây phạm vi được đề xuất dựa vào các kỹ thuật hình học tính toán. Nghiên cứu cũng đề xuất phương pháp tối ưu thuật toán tìm trên cây phạm vi để khắc phục trường hợp suy biến cho tập điểm đầu vào có cùng tọa độ x hoặc tọa độ y bằng phương pháp tập điểm chung. Hướng nghiên cứu tiếp theo của bài báo sẽ tiếp tục tập trung vào các phương pháp tìm kiếm trên miền trực giao trên cơ sở dữ liệu tương ứng được trình bày trong nghiên cứu này để phát triển bài toán chuyển đổi các

bản ghi trong cơ sở dữ liệu thành tập các điểm trong không gian nhiều chiều mà đáp ứng được nhu cầu ngày càng tăng về tốc độ trong truy vấn thông tin hiện nay.

Tài liệu tham khảo

1. Chan, Timothy M. and Huang, Zhengcheng. (2021). Dynamic Colored Orthogonal Range Searching. *Schloss Dagstuhl-Leibniz-Zentrum Informatik,* vol. 204, pp.1-13.
2. Eldawy A, Li Y, Mokbel MF, Janardan R. (2013). CG_Hadoop: computational Geometry in mapreduce. *Proceedings of the international conference on advances in geographic information systems*, ACM SIGSPATIAL, pp. 284–293.
3. Francesca Rossi, Peter van Beek, Toby Walsh. (2006). In *Handbook of Constraint Programming* (pp. 1-20). Elsevier Science.
4. H, Chen W. (2016). GPU-accelerated blind and robust 3D mesh watermarking by geometry image. *Int J Multimedia Tools Appl,* vol. 75, no. 16, pp.10077–10096.
5. Heshan Da, Natasha Alechins. Michael Jackson and Glen Hart. (2017). A Method for Maching Caned arced and Authoritative Geospatial Data. Transactions in GIS, 212, pp 406427. 10.1111/tgis.12210.
6. Jacob E. Goodman, Joseph O'Rourke, and Csaba D. Tóth. (2017). *Handbook of Discrete and Computational Geometry* (pp. 3-10). CRC Press LLC
7. Kazuki Ishiyama, Kunihiko Sadakane. (2017). A succinct data structure for multidimensional orthogonal range searching. Proceedings of Data Compression Conference, IEEE, pp. 270–279. 10.1109/DCC.2017.47
8. Kazuki Ishiyama, Kunihiko Sadakane. (2017). Practical space-efficient data structures for high-dimensional orthogonal range searching, *Proceedings of the 10th International Conference on Similarity Search and Applications, SISAP*, Springer, pp. 234–246.
9. Kazuki Ishiyama, Kunihiko Sadakane. (2020). Compact and succinct data structures for multidimensional orthogonal range searching. *Information and Computation, 273. https://doi.org/10.1016/j.ic.2020.104519*
10. Larsen, K.G., Williams, R. (2017). Faster online matrix-vector multiplication. *In: Proceedings of the 28th ACM–SIAM Symposium on Discrete Algorithms (SODA'17)*, SIAM, Philadelphia, 2017, pp. 2182–2189.
11. Le Gall, F., Urrutia, F. (2018). Improved rectangular matrix multiplication using powers of the Coppersmith–Winograd tensor. In: *Proceedings of the 29th ACM–SIAM Symposium on Discrete Algorithms (SODA'18)*, SIAM, Philadelphia, 2018, pp. 1029–1046.
12. Lê Thị Thuấn. (2023). Nghiên cứu các kỹ thuật của hình học tính toán cho thuật toán tìm kiếm phạm vi hai chiều hỗ trợ bài toán truy vấn cơ sở dữ liệu. *TNU Journal of Science and Technology*, 228(07), pp. 20-27.
13. Mark de Berg, Otfried Cheong, Marc van Kreveld, Mark Overmars. (2008). *Computational Geometry* (pp.1-40), Springer.
14. McKenney M, Frye R, Dellamano M, Anderson K, Harris J. (2017). Multi-core parallelism for plane sweep algorithms as a foundation for GIS operations. *GeoInformatica 21(1),* pp.151–174.