Tiểu luận cuối kỳ DSA 2023

Nhóm 9

R Tree: Theory & Applications

Các tác giả:

Harito Team: Github Link Project

Phạm Ngọc Hải \_ 21002139

Lương Đức Anh \_ 21002117

Nguyễn Văn Thắng \_ 21002175

# Lời mở đầu

Tiểu luận bao gồm chín chương được tổ chức thành bốn phần. Chúng tôi đã cố gắng giữ cho mỗi chương độc lập để mang lại sự linh hoạt tối đa cho việc đọc.

Phần đầu của tiểu luận bao gồm một số vấn đề cơ bản và bao gồm ba chương. Chương 1 là chương mở đầu. Trong chương này, chúng tôi **giới thiệu** ngắn gọn về hoàn cảnh ra đời, lĩnh vực ứng dụng, đồng thời trình bày các phương thức truy cập của R-tree ban đầu, là gốc rễ của các phương thức truy cập được trình bày trong tiểu luận. Chương 2 dành cho việc mô tả **các biến thể động** hứa hẹn nhất của R-tree. Các phương pháp này hỗ trợ chèn, xóa, cập nhật và do đó có thể được sử dụng hiệu quả trong môi trường động. **Biến thể tĩnh** của R-tree được đưa ra trong Chương 3. Các biến thể này được tối ưu hóa khi xem xét rằng tập dữ liệu được sắp xếp đã được cung cấp trước.

Phần thứ hai của tiểu luận bao gồm hai chương và đề cập đến các kỹ thuật xử lý truy vấn đã được đề xuất để vận hành với cây R. Chương 4 nghiên cứu **các loại truy vấn cơ bản** như truy vấn phạm vi, truy vấn lân cận gần nhất và truy vấn nối không gian. Mỗi phương pháp được nghiên cứu chi tiết và thuật toán tương ứng được đưa ra dưới dạng mã giả thích hợp. Chương 5 khám phá **các loại truy vấn phức tạp** hơn như truy vấn phạm vi phân loại, nối không gian nhiều chiều, truy vấn cặp gần nhất, xử lý gia tăng và kỹ thuật truy vấn gần đúng. Các truy vấn này được đặc trưng bởi chi phí tính toán cao hơn và độ phức tạp lớn hơn so với các truy vấn cơ bản và do đó được đề cập riêng.

Sự thích ứng của cây R với các lĩnh vực ứng dụng hiện đại được thảo luận trong phần thứ ba, bao gồm hai chương. Chương 6 nghiên cứu ứng dụng của các phương pháp truy cập dạng cây R cho các **hệ cơ sở dữ liệu không gian thời gian**. Đặc điểm cơ bản của các hệ thống này là chúng xử lý thông tin thời gian bên cạnh các thuộc tính không gian của các đối tượng. Chương 7 thảo luận về việc sử dụng cây R trong **cơ sở dữ liệu đa phương tiện, kho dữ liệu và các nhiệm vụ khai thác dữ liệu**. Việc khai thác R-tree bởi các miền nói trên đã chứng tỏ rất hứa hẹn về các thuật toán và kỹ thuật xử lý truy vấn nhanh hơn, có tính đến độ phức tạp của các đối tượng và các hoạt động tính toán chuyên sâu cần thiết.

Phần cuối của tiểu luận là 1 số chia sẻ của Harito Team chúng tôi về **cách triển khai các cấu trúc dữ liệu và thuật toán đã nêu bằng ngôn ngữ Java**. Để xem chi tiết về những gì chúng tôi đã triển khai có thể tham khảo qua link Github của chúng tôi.

# 

# Mục lục:

[Lời mở đầu](#_3k2whw8ulr5h)

[Mục lục:](#_zdiu1a6on00p)

[Part I. Khái niệm cơ bản.](#_6ei2bw8ics4l)

[1. Giới thiệu:](#_vxceat81br49)

[1.1. Hoàn cảnh ra đời:](#_9313c89rc52u)

[1.2. Cấu trúc & Thuật toán trên R Tree nguyên bản:](#_iqr5dkub5emv)

[1.2.1. Cấu trúc:](#_h8wc4kteziv8)

[1.2.2. Thuật toán:](#_2mjmvqx9mvkx)

[1.2.2.1. Tìm kiếm:](#_1hx18jatn39a)

[1.2.2.2. Chèn:](#_nl4r9s44o23a)

[1.2.2.3. Xóa:](#_sv9zbtd7z5b9)

[1.3. Tổng kết:](#_8xoik8wt6kty)

[2. Các phiên bản động của R Tree:](#_gzw8o4gt23ql)

[2.1. The R+ -tree:](#_kp33uz58vzkq)

[2.2. The R\*-tree:](#_6c4kaise3gs3)

[2.3. The Hilbert R-tree:](#_gf8vf7kpsq58)

[2.4. Linear Node Splitting:](#_rcp4qisx0dr2)

[2.5. Optimal Node Splitting:](#_rf0y4x2e8zk7)

[2.6. Branch Grafting:](#_o3p2cxz40e0v)

[2.7. Compact R-trees:](#_ldwn8we7rlr4)

[2.8. cR-trees:](#_fcdunnci7me7)

[2.9. Deviating Variations:](#_vqvrnmjyowsc)

[2.9.1. PR-trees:](#_3vhjvl9ol5wi)

[2.9.2. LR-trees:](#_mk7vi9qnr7oy)

[2.10. Summary:](#_4b33pu42qjuf)

[3. Các phiên bản tĩnh của R Tree:](#_4fo1j376ronv)

[3.1. The Packed R-tree:](#_eb0o9gjs5lxi)

[3.2. The Hilbert Packed R-tree:](#_7gqs3sdga39b)

[3.3. The STR R-tree:](#_djg8gpmzdx7q)

[3.4. Top-Down Packing Techniques:](#_fsugxbmhz6c7)

[3.5. Small-Tree-Large-Tree and GBI:](#_7l02o19yb2g6)

[3.6. Bulk Insertion by Seeded Clustering:](#_mn14pj4k0i7e)

[3.7. The Buﬀer R-tree:](#_hbfxktfmv5db)

[3.8. R-tree with Low Stabbing Number:](#_2bvcwzh90l0m)

[3.9. Merging R-trees:](#_z69zg51x5lwy)

[3.10. Summary:](#_ar2e012op3s1)

[Part II. Vấn đề xử lý truy vấn.](#_idlw99muo6ls)

[4. Kỹ thuật xử lý truy vấn cơ bản:](#_id8xjlvjfzpy)

[4.1. Xử lý hai bước:](#_f5m2djqaque5)

[4.2. Truy vấn phạm vi và truy vấn tô pô:](#_hwbv064zttem)

[4.3. Truy vấn Nearest-Neighbor:](#_w40uflhfym99)

[4.3.1. Thuật toán rẽ nhánh và giới hạn:](#_olj6a7srjb5k)

[4.3.2. Cải thiện thuật toán gốc:](#_mojapq36mbik)

[4.3.3. Tim kiếm Nearest-Neighbor gia tăng:](#_hi5mkuyijy0v)

[4.3.4. So sánh các thuật toán Nearest-Neighbor:](#_eovqvt4o3q6d)

[4.4. Truy vấn nối không gian:](#_kblt3etbmjes)

[4.4.1. Thuật toán dựa trên duyệt theo độ sâu:](#_8x7fi82ka8b)

[4.4.2. Thuật toán dựa trên duyệt theo chiều rộng:](#_2sne2v9luyrr)

[4.4.3. Nối giữa tập dữ liệu được lập chỉ mục trên R-Tree và không được lập chỉ mục trên R-Tree:](#_dkek03lneopg)

[4.5. Tóm tắt:](#_u82ersalgxb)

[5. Xử lý các truy vấn phức tạp hơn:](#_ydmmg5u2oef1)

[5.1. Truy vấn phạm vi phân loại:](#_62wayqfl8hqu)

[5.2. Truy vấn Nearest-Neighbor bị ràng buộc và đảo ngược:](#_oc5xyc2m8ald)

[5.2.1. Nearest-Neighbors đảo ngược:](#_kf00fdb2ersi)

[5.2.2. Tìm kiếm Nearest-Neighbor bị ràng buộc tổng quát:](#_un5cbv9o1p02)

[5.3. Truy vấn nối không gian đa chiều:](#_8sxqyu9wywfn)

[Part III. R-Trees trong các ứng dụng hiện đại](#_as3fw7vfc2co)

[6. R-trees trong cơ sở dữ liệu không gian:](#_34kl3ykcqdii)

[6.1. Mở đầu:](#_87nmsnsx6g79)

[6.2. RT-tree:](#_ow1xdboe1pmj)

[6.3. R-tree 3 chiều:](#_ocq37hroslgb)

[6.4. R-tree 2 + 3:](#_wvf12325siq1)

[6.5. HR-tree:](#_m3fuega7s216)

[6.6. Rst-tree:](#_w4xrea7fbxqo)

[6.7. PPR-tree:](#_5ky37p10twru)

[6.8. MV3R-tree](#_4vd2z2s2izyj)

[6.9. TB-tree:](#_5cgvy0tlgg0w)

[6.10. Scalable and Efficient Trajectory Index (SETI)](#_45c5j7z9mzte)

[6.11. Q+R-tree](#_kqczs4cl517d)

[6.12. FRN-tree và MON-tree](#_g2me9qea8dtp)

[6.13. R-tree tham số hoá thời gian](#_da28clyy5cqe)

[6.14. VCI R-tree](#_21aoqmbrybpb)

[6.15. Tổng kết](#_6cjbqslvhg3l)

[7. R-tree trong đa phương tiện, lưu trữ và khai thác](#_yqyttz43mwfz)

[7.1. R-tree trong cơ sở dữ liệu đa phương tiện](#_wbo2yw65kn2q)

[7.1.1. Lập chỉ mục đa phương tiện chung (GEMINI)](#_tnjwhdp0qs7)

[7.1.2. Phương thức truy cập chiều cao](#_r3mgp9krr0tt)

[7.1.3 R-tree và mô hình Markov ẩn trong truy xuất âm nhạc](#_lmvyoahljurf)

[7.1.4. R-tree và bản đồ tự tổ chức](#_nozywfnp7m3z)

[7.2. R-tree trong lưu trữ và khai thác dữ liệu](#_5wr83edsqgq2)

[7.3. Tổng kết](#_4b6xybaxda8h)

# Part I. Khái niệm cơ bản.

## 1. Giới thiệu:

### 1.1. Hoàn cảnh ra đời:

Những năm 1980 là thời kỳ chấp nhận rộng rãi của các mô hình quan hệ, nhưng đồng thời mô hình quan hệ là không đủ để lưu trữ các ứng dụng mới nổi. Đa phương tiện, [CAD/CAM](https://vcc-tech.vn/tim-hieu-ve-cong-nghe-cad-cam-cnc.html) - thiết kế sản phẩm sản xuất hàng loạt, xử lý dữ liệu không gian như hệ thống thông tin địa lý (GIS), các ứng dụng đồ họa, y tế và khoa học, hệ thống tìm kiếm, v.v. là một số ví dụ. Một trong những lý do cho sự thiếu sót của các hệ thống quan hệ là không có khả năng xử lý các loại dữ liệu mới với [B-tree (balance tree)](https://en.wikipedia.org/wiki/B-tree). Cụ thể hơn, B-tree được thiết kế để xử lý dữ liệu chữ và số (nghĩa là một chiều), như số nguyên, ký tự và chuỗi, trong đó có thể xác định quan hệ thứ tự. Một số biến thể cây B mới đã xuất hiện để xử lý dữ liệu hướng đối tượng. Chủ yếu, những cấu trúc này có mục tiêu nhằm mục đích lưu trữ dữ liệu của các hệ thống phân cấp đối tượng trong một cấu trúc duy nhất. Tuy nhiên, những nỗ lực này có khả năng áp dụng hạn chế. Trong hoàn cảnh đó, [R-tree](https://en.wikipedia.org/wiki/R-tree) được đề xuất bởi **Guttman vào năm 1984**, nhằm mục đích xử lý dữ liệu hình học, chẳng hạn như điểm, đoạn thẳng, bề mặt, thể tích và siêu thể tích trong không gian nhiều chiều.

Ngày nay, cơ sở dữ liệu không gian và hệ thống thông tin địa lý đã là một lĩnh vực trưởng thành, cơ sở dữ liệu không gian thời gian và thao tác của các điểm quỹ đạo chuyển động vẫn được nghiên cứu rộng rãi và cuối cùng là cơ sở dữ liệu hình ảnh và đa phương tiện có thể xử lý các loại dữ liệu mới, chẳng hạn như hình ảnh, giọng nói, âm nhạc hoặc video. Tất cả chúng đều ứng dụng dựa vào R-tree như một công cụ cần thiết cho lưu trữ và truy xuất. Ứng dụng ban đầu thúc đẩy Guttman trong nghiên cứu là thiết kế [VLSI](https://en.wikipedia.org/wiki/Very_Large_Scale_Integration) (tức là, làm thế nào để trả lời một cách hiệu quả liệu một khoảng trống đã được bao phủ bởi một con chip hay chưa). Dần dần, xử lý hình chữ nhật nhanh chóng tìm thấy các ứng dụng trong địa lý và nói chung, dữ liệu không gian, bao gồm cả hệ thống thông tin địa lý (các tòa nhà, sông, thành phố, v.v.), hình ảnh hoặc hệ thống truy xuất video / âm thanh (sự giống nhau của các đối tượng trong không gian gốc hoặc không gian đặc trưng chiều cao), chuỗi thời gian và cơ sở dữ liệu thời gian (thời gian - khoảng thời gian chỉ là đối tượng 1D), v.v. Do đó, có thể lập luận rằng cây R được tìm thấy ở khắp mọi nơi.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Cách giới thiệu khác:

Trong quá khứ, người ta chỉ cần lưu trữ và truy vấn các dữ liệu đơn chiều vì các hệ thống và ứng dụng không đòi hỏi nhiều thông tin về vị trí và mối quan hệ giữa các đối tượng trong không gian đa chiều. Các dữ liệu đơn chiều là các dữ liệu có thể biểu diễn trên một trục đơn, ví dụ như các giá trị số nguyên, số thực, ký tự, boolean, thời gian hoặc địa chỉ.

Các kiểu dữ liệu phổ biến được sử dụng để lưu trữ và truy vấn các dữ liệu đơn chiều bao gồm:

1. Mảng (Array): Lưu trữ các giá trị dữ liệu trên một trục đơn, và cho phép truy cập nhanh chóng đến các giá trị dữ liệu bằng cách sử dụng chỉ số.

2. Danh sách liên kết (Linked List): Lưu trữ các giá trị dữ liệu trên một trục đơn trong các nút của danh sách liên kết, và cho phép truy cập tuyến tính đến các giá trị dữ liệu.

3. Bảng băm (Hash Table): Lưu trữ các giá trị dữ liệu trên một trục đơn dựa trên quy tắc băm, và cho phép truy cập nhanh chóng đến các giá trị dữ liệu bằng cách sử dụng khóa.

4. Cây tìm kiếm nhị phân (Binary Search Tree): Lưu trữ các giá trị dữ liệu trên một trục đơn trong các nút của cây tìm kiếm nhị phân, và cho phép truy cập nhanh chóng đến các giá trị dữ liệu bằng cách sử dụng thuật toán tìm kiếm nhị phân.

Các kiểu dữ liệu này thường được sử dụng để lưu trữ và truy vấn các dữ liệu đơn chiều trong các ứng dụng khác nhau, bao gồm hệ thống quản lý cơ sở dữ liệu, phân tích dữ liệu và nhiều lĩnh vực khác.

Array, linked list, hash table và binary tree là các kiểu cấu trúc dữ liệu phổ biến được sử dụng để lưu trữ và truy vấn dữ liệu đơn chiều. Tuy nhiên, chúng cũng có thể được sử dụng để lưu trữ và truy vấn dữ liệu đa chiều, tuy nhiên hiệu quả của chúng sẽ bị giảm xuống so với khi sử dụng các cấu trúc dữ liệu được thiết kế đặc biệt cho dữ liệu đa chiều như R-tree hoặc quadtree. Dưới đây là cách sử dụng các kiểu cấu trúc dữ liệu này để lưu trữ và truy vấn dữ liệu đa chiều:

1. Array: Mảng có thể được sử dụng để lưu trữ các giá trị dữ liệu trong một không gian đa chiều bằng cách sử dụng một mảng đa chiều. Để truy vấn dữ liệu trong một mảng đa chiều, bạn có thể sử dụng các chỉ số để truy cập các giá trị dữ liệu tại một vị trí cụ thể trong mảng.

2. Linked list: Linked list có thể được sử dụng để lưu trữ dữ liệu đa chiều bằng cách sử dụng các nút trong danh sách liên kết để lưu trữ các giá trị dữ liệu. Mỗi nút trong danh sách liên kết có thể chứa một giá trị dữ liệu và một con trỏ đến nút tiếp theo trong danh sách. Để truy vấn dữ liệu trong danh sách liên kết, bạn có thể sử dụng các phương pháp tìm kiếm tương tự như khi sử dụng mảng.

3. Hash table: Hash table có thể được sử dụng để lưu trữ và truy vấn dữ liệu đa chiều bằng cách sử dụng chức năng băm để ánh xạ các giá trị dữ liệu vào một bảng băm. Để truy vấn dữ liệu trong một hash table, bạn có thể sử dụng các khóa để tìm kiếm các giá trị dữ liệu tương ứng.

4. Binary tree: Cây tìm kiếm nhị phân có thể được sử dụng để lưu trữ và truy vấn dữ liệu đa chiều bằng cách sử dụng các nút trong cây để lưu trữ các giá trị dữ liệu. Mỗi nút trong cây có thể chứa một giá trị dữ liệu và hai con trỏ đến các cây con trái và phải. Để truy vấn dữ liệu trong một cây tìm kiếm nhị phân, bạn có thể sử dụng các thuật toán tìm kiếm nhị phân để tìm kiếm các giá trị dữ liệu tương ứng.

Tuy nhiên, các kiểu cấu trúc dữ liệu này thường không đủ hiệu quả để lưu trữ và truy vấn các dữ liệu đa chiều phức tạp trong thực tế. Trong nhiều trường hợp, các kiểu dữ liệu đa chiều như R-tree hoặc quadtree được sử dụng để biểu diễn dữ liệu đa chiều một cách hiệu quả hơn.

### 1.2. Cấu trúc & Thuật toán trên R Tree nguyên bản:

#### 1.2.1. Cấu trúc:

R-tree là [cấu trúc dữ liệu phân cấp](https://butchiso.com/2013/06/cau-truc-du-lieu-phan-cap-va-ung-dung.html) dựa trên [B+ - trees](https://en.wikipedia.org/wiki/B%2B_tree). Chúng được sử dụng cho tổ chức động của một tập hợp d-chiều các đối tượng hình học đại diện cho chúng bằng hình chữ nhật tối thiểu giới hạn d-chiều (minimum bounding d-dimensional rectangles - MBRs).

* Mỗi node trong của R-tree tương ứng với MBR giới hạn của nó.
* Mỗi node lá của R-tree có chứa con trỏ tới các đối tượng cơ sở dữ liệu thay vì con trỏ tới các node con.

Cần lưu ý rằng các MBR bao quanh các node khác nhau có thể chồng lên nhau nhau. Ngoài ra, một MBR có thể được đưa vào (theo nghĩa hình học) trong nhiều node (tức có thể có nhiều MBR khác lớn hơn bao quanh, chứa MBR đó), nhưng nó chỉ có thể được liên kết với một trong số chúng (tức không được có nhiều hơn 1 node cha) - liên kết đến MBR nhỏ nhất bao quanh nó.

Điều này có nghĩa rằng một tìm kiếm không gian có thể truy cập nhiều node (nhiều MBR lớn hơn) trước khi xác nhận sự tồn tại của một MBR. Ngoài ra, dễ thấy rằng việc biểu diễn các đối tượng hình học thông qua MBR của chúng có thể dẫn đến báo động sai. Ví dụ trường hợp hai đa giác không cắt nhau, nhưng MBR của chúng thì có. Để giải quyết các cảnh báo sai này, các đối tượng đang xét sẽ phải được kiểm tra. Do đó, R-tree đóng vai trò là một cơ chế lọc để giảm việc kiểm tra trực tiếp các đối tượng hình học.

**Định nghĩa**: Một cây R bậc (m, M) có các đặc điểm sau:

* Mỗi **leaf node** (trừ khi nó là root node) có thể lưu trữ **tối đa M** [**entry**](https://dictionary.cambridge.org/vi/dictionary/english/entry), và **tối thiểu là m entry** (với m ≤ M / 2). Mỗi **entry có dạng (mbr, oid)**, sao cho mbr là MBR chứa đối tượng trong không gian (liên kết đến node cha) và oid là key định danh của đối tượng.
* Số entry mà mỗi **node bên trong - internal node** cũng có thể lưu trữ nằm trong khoảng m và M (với m ≤ M / 2). Những **entry có dạng (mbr, p)**, trong đó p là liên kết đến node con và mbr là liên kết đến node cha.
* Số lượng entry tối thiểu được phép trong root node là 2, trừ khi đó là một leaf node (trong trường hợp này chỉ có duy nhất 1 điểm data trong không gian).
* Tất cả các leaf node của R-tree đều ở cùng cấp độ.

**Nhận xét**:

* Từ định nghĩa, suy ra R-tree là [cây cân bằng chiều cao](https://www.geeksforgeeks.org/introduction-to-height-balanced-binary-tree/). Như đã đề cập, nó bao gồm sự khái quát hóa cấu trúc B+ -tree cho nhiều chiều. R-tree là [cấu trúc dữ liệu động](https://www.geeksforgeeks.org/static-data-structure-vs-dynamic-data-structure/), tức việc tổ chức lại toàn cục cây là không bắt buộc phải xử lý với các thao tác như chèn hoặc xóa.
* Cho một R-tree lưu trữ N hình chữ nhật. Trong trường hợp này giá trị tối đa cho **chiều cao h** của nó là:

$$h\_{max} = \lceil\log\_m(N)\rceil - 1$$

* Số node tối đa có thể được tính bằng cách tính tổng số node tối đa có thể có trên mỗi cấp độ. Con số này xuất hiện khi tất cả các node chứa số lượng mục nhập tối thiểu được phép (m) là:

$$\sum\_{i=1}^{h\_{max}}\left\lceil N/m^{i} \right\rceil = \left\lceil N/m \right\rceil + \left\lceil N/m^{2} \right\rceil + ... + 1$$

#### => tại sao r tree là phổ biến nhất để lưu trữ và truy vấn dữ liệu đa chiều

Nhiều cấu trúc đã được đề xuất để xử lý dữ liệu đa chiều: Cell methods, Quad trees, kd trees, KDB Trees. Nhưng vì những lý do khác nhau, các phương pháp này đã được chứng minh là không phù hợp (không tốt cho cấu trúc động, chỉ hữu ích cho dữ liệu điểm...) . Trong số các phương thức chức năng, chúng ta sẽ xử lý cụ thể cấu trúc cây R, biểu diễn các đối tượng dữ liệu theo các khoảng trong một số chiều.

R-tree là một trong những cấu trúc dữ liệu phổ biến nhất để lưu trữ và truy vấn dữ liệu đa chiều vì có một số ưu điểm quan trọng như sau:

1. Hiệu quả trong việc truy vấn: R-tree cho phép truy vấn nhanh chóng các đối tượng dựa trên vị trí không gian của chúng. Trong R-tree, các đối tượng được phân bố trong các vùng con của không gian dữ liệu, và các vùng con này được tổng hợp lại thành các vùng lớn hơn dựa trên một số tiêu chí. Do đó, việc truy vấn dữ liệu sẽ nhanh chóng hơn so với việc duyệt qua toàn bộ tập dữ liệu.

2. Hỗ trợ truy vấn dữ liệu đa chiều: R-tree có thể được sử dụng để lưu trữ và truy vấn dữ liệu đa chiều, bao gồm cả không gian hai chiều và ba chiều.

3. Khả năng mở rộng: R-tree có thể được mở rộng dễ dàng để hỗ trợ các tập dữ liệu lớn hơn. Khi tập dữ liệu lớn hơn, R-tree có thể được chia thành các cây con để tăng tốc độ truy vấn.

4. Độ phức tạp thấp: R-tree có độ phức tạp thấp trong việc truy vấn và lưu trữ dữ liệu, giúp giảm thiểu tài nguyên máy tính cần thiết cho việc xử lý dữ liệu.

Những ưu điểm này làm cho R-tree trở thành một trong những cấu trúc dữ liệu phổ biến nhất để lưu trữ và truy vấn dữ liệu đa chiều trong nhiều ứng dụng thực tế, bao gồm hệ thống thông tin địa lý (GIS), truy vấn hình ảnh và âm thanh, truy vấn tài liệu, và nhiều lĩnh vực khác.

#### 1.2.2. Thuật toán:

##### 1.2.2.1. Tìm kiếm:

Bài toán giả thiết có hình chữ nhật P, chúng ta có thể tạo truy vấn sau: **tìm tất cả các hình chữ nhật dữ liệu bị cắt bởi P**. Điều này được biểu thị dưới dạng truy vấn phạm vi. Thuật toán xử lý các truy vấn phạm vi trong cây R đối với một entry E, E.mbr biểu thị MBR tương ứng và E.pchild trỏ tương ứng đến cấp độ tiếp theo. Nếu E là một chiếc lá, thì E.pchild chỉ định danh đối tượng tương ứng (oid).

Thuật toán tìm kiếm phân tách ([partition search algorithm](https://ieeexplore.ieee.org/document/5293260)) có thể giảm thiểu số lượng khối cần phải truy xuất để tìm kiếm một đối tượng cụ thể. Các thuật toán khác như tìm kiếm theo phạm vi (range search algorithm) hoặc kiểm tra sự chồng chéo (overlap test algorithm) giữa các đối tượng trong không gian đa chiều cũng được thực hiện trên R-Tree.

Sau đây là mã giả cho thuật toán giải quyết vấn đề trên:

-------------------------------------------------------------------------------------------------------------------------------

Algorithm Range Search (TypeNode RN , TypeRegion Q)

/\* Finds all rectangles that are stored in an R-tree with root node RN, which are intersected by a query rectangle Q. Answers are stored in the set A \*/

if (RN is not a leaf node) {

examine each entry e of RN to ﬁnd those e.mbr that intersect Q

foreach such entry e call RangeSearch(e.ptr,Q)

} else // RN is a leaf node {

examine all entries e and ﬁnd those for which e.mbr intersects Q

add these entries to the answer set A

}

-------------------------------------------------------------------------------------------------------------------------------

Sau đây là mã Java thực thi thuật toán:

…

Chúng tôi lưu ý rằng các hình chữ nhật được tìm thấy bằng cách tìm kiếm phạm vi được cấu thành các hình chữ nhật đang xét của bước lọc. Các đối tượng hình học thực tế giao nhau bởi hình chữ nhật truy vấn phải được tìm thấy trong một bước sàng lọc bằng cách truy xuất đối tượng của các hình chữ nhật con (bị chứa) và kiểm tra giao điểm của chúng (tức là đệ quy tới leaf node để xác nhận giao điểm của hình chữ nhật đang xét).

##### 1.2.2.2. Chèn:

Bài toán các phần chèn vào cây R được xử lý tương tự như các phần chèn vào cây B+-tree. Đặc biệt, cây R được duyệt qua để định vị một lá thích hợp để chứa entry mới. Entry mới được chèn vào lá tìm thấy và sau đó tất cả các node trong đường dẫn từ gốc đến lá đó được cập nhật tương ứng. Trường hợp lá tìm được không chứa được entry mới do đã đầy (đã có M tối đa các entry) thì sau đó nó được chia thành hai node. Lúc này thì thao tác tách node được thực hiện. Lưu ý tách trong R-tree khác với tách trong B+- cây, bởi vì nó xem xét các tiêu chí khác nhau.

Sau đây là mã giả cho thuật toán giải quyết vấn đề trên:

-------------------------------------------------------------------------------------------------------------------------------

Algorithm Insert(TypeEntry E, TypeNode RN )

/\* Inserts a new entry E in an R-tree with root node RN \*/

Traverse the tree from root RN to the appropriate leaf. At each level, select the node, L, whose MBR will require the minimum area enlargement to cover E.mbr;

In case of ties, select the node whose MBR has the minimum area;

if (the selected leaf L can accommodate E) {

Insert E into L;

Update all MBRs in the path from the root to L, so that all of them cover E.mbr;

} else // L is already full {

Let E be the set consisting of all L’s entries and the new entry E;

Select as seeds two entries e1 , e2 ∈ E, where the distance between e1 and e2 is the maximum among all other pairs of entries from E;

Form two nodes, L1 and L2 , where the ﬁrst contains e1 and the second e2;

Examine the remaining members of E one by one and assign them to L1 or L2, depending on which of the MBRs of these nodes will require the minimum area enlargement so as to cover this entry;

if (a tie occurs) {

Assign the entry to the node whose MBR has the smaller area;

}

if (a tie occurs again) {

Assign the entry to the node that contains the smaller number of entries;

}

if (during the assignment of entries, there remain λ entries to be assigned and the one node contains m − λ entries) {

Assign all the remaining entries to this node without considering the aforementioned criteria;

/\* so that the node will contain at least m entries \*/

}

Update the MBRs of nodes that are in the path from root to L, so as to cover L1 and accommodate L2;

Perform splits at the upper levels if necessary;

In case the root has to be split, create a new root;

Increase the height of the tree by one;

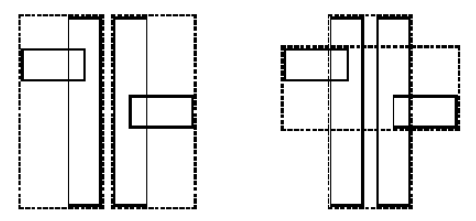
}

-------------------------------------------------------------------------------------------------------------------------------

Sau đây là mã Java thực thi thuật toán:

…

Thuật toán chèn nói trên sử dụng cái gọi là [thuật toán phân chia tuyến tính - linear split algorithm](https://stackoverflow.com/questions/17289393/difference-between-quadratic-split-and-linear-split) (nó có độ phức tạp thời gian tuyến tính). Mục tiêu của thuật toán phân tách là giảm thiểu xác suất gọi cả hai node đã tạo (L1 và L2). Vì với cùng một truy vấn, thuật toán phân chia tuyến tính cố gắng đạt được mục tiêu này bằng cách giảm thiểu (tìm MIN) tổng diện tích của hai node được tạo. Ví dụ về sự phân tách xấu và tốt được đưa ra trong Hình 1.6. Ở phần bên trái của hình, sự phân chia không tốt vì MBR của các node thu được có diện tích lớn hơn nhiều so với diện tích được mô tả ở phần bên phải của hình.



Tuy nhiên, thuật toán phân tách tuyến tính là một trong ba giải pháp thay thế cho xử lý phân tách cho việc chèn được Guttman đề xuất. Hai cái còn lại có độ phức tạp bậc hai và hàm mũ. Ba lựa chọn thay thế này được tóm tắt như sau:

[Linear Split](https://stackoverflow.com/questions/17289393/difference-between-quadratic-split-and-linear-split). Chọn hai đối tượng làm hạt giống cho hai node, trong đó các đối tượng này càng xa nhau càng tốt. Sau đó xem xét từng đối tượng còn lại trong một thứ tự ngẫu nhiên và gán nó cho với node chỉ yêu cầu MBR tương ứng của nó cần được mở rộng nhỏ nhất.

[Quadratic Split](https://stackoverflow.com/questions/17289393/difference-between-quadratic-split-and-linear-split). Chọn hai đối tượng làm hạt giống cho hai node, trong đó các đối tượng này nếu được đặt cùng nhau sẽ tạo ra khoảng cách Euclid càng lớn càng tốt. Sau đó, cho đến khi không còn đối tượng nào nữa, hãy chèn đối tượng mà chênh lệch khoảng cách Euclid nếu được gán cho từng node trong số hai node được tối đa hóa trong node yêu cầu mở rộng MBR tương ứng của nó ít hơn.

[Exponential Split](https://stackoverflow.com/questions/22158394/tree-with-exponential-split-factor). Tất cả các nhóm có thể được kiểm tra kỹ lưỡng và nhóm tốt nhất được chọn liên quan đến việc giảm thiểu việc mở rộng MBR, tức ta sẽ quét tuyến tính mọi nhóm MBR đang có, sau đó tính tổng khoảng cách Euclid của node sẽ thêm vào với từng node có trong MBR đó, MBR được chọn để thêm node mới vào là MBR mà tại đó tổng khoảng cách Euclid thu được là MIN.

Guttman đề xuất sử dụng **Quadratic Split Algorithm** như một sự thỏa hiệp tốt để đạt được hiệu suất truy xuất hợp lý.

##### 1.2.2.3. Xóa:

Sau đây là mã giả cho việc xoá điểm điểm dữ liệu entry E khỏi cây có gốc là RN.

-------------------------------------------------------------------------------------------------------------------------------

Algorithm Delete (TypeEntry E, TypeNode RN )

/\* Deletes an entry E from an R-tree with root node RN \*/

if (RN is a leaf node) {

search all entries of RN to ﬁnd E.mbr

} else // RN is an internal node {

Find all entries of RN that cover E.mbr

Follow the corresponding subtrees until the leaf L that contains E is found

Remove E from L

}

Call algorithm CondenseTree(L) /\*Bên dưới \*/

if (the root has only one child /\* and it is not a leaf \*/) {

Remove the root

Set as new root its only child

}

-------------------------------------------------------------------------------------------------------------------------------

Algorithm CondenseTree(TypeNode L)

/\* Given is the leaf L from which an entry E has been deleted. If after the deletion of E, L has fewer than m entries, then remove entirely leaf L and reinsert all its entries. Updates are propagated upwards and the MBRs in the path from root to L are modiﬁed (possibly become smaller) \*/

Set X = L

Let N be the set of nodes that are going to be removed from the tree (initially, N is empty)

while (X is not the root) {

Let P arentX be the father node of X

Let Entry of X be the entry of Parent of X that corresponds to X

if (X contains less than m entries) {

Remove EX from ParentX

Insert X into N

}

if (X has not been removed) {

Adjust its corresponding MBR EX .mbr, so as to enclose all rectangles in X

/\* Entry of X .mbr may become smaller \*/

}

Set X = Parent of X

}

Reinsert all the entries of nodes that are in the set N

-------------------------------------------------------------------------------------------------------------------------------

**Nhận xét**: Trong R-tree, khi một nút có ít hơn m phần tử, nút đó được coi là "underflow node" và có thể được xử lý bằng cách chèn lại các phần tử đã xoá vào cây.

Trong B+ -tree, khi có hai nút anh em cùng cấp không đủ số lượng phần tử, ta có thể nối hai nút này lại với nhau. Tuy nhiên, trong R-tree, đối với dữ liệu đa chiều, việc nối hai nút anh em lại với nhau có thể không đảm bảo tính liên tục của dữ liệu và không phù hợp trong mọi trường hợp. Lý do cụ thể hơn như sau:

- Chèn lại vào cây đạt được kết quả tương tự như việc hợp nhất node. Ngoài ra, vì số lần truy cập đĩa trong quá trình xóa là rất quan trọng đối với hiệu suất khi xóa, mà cần phải lưu ý rằng các node được yêu cầu trong quá trình chèn lại đã có sẵn trong bộ nhớ đệm - chúng đã được truy xuất trong quá trình tìm kiếm entry thoả mãn để xóa. Như vậy đạt được kết quả tương tự mà việc chèn lại sẽ có hiệu suất nhanh hơn việc hợp nhất node.

- Như đã mô tả, thuật toán chèn cố gắng duy trì chất lượng tốt của cây trong quá trình thực hiện truy vấn. Do đó, sẽ hợp lý hơn khi sử dụng chèn lại, vì chất lượng của cây có thể giảm sau vài lần xóa. Như vậy, việc chèn lại các phần tử đã xoá vào cây được ưu tiên hơn, vừa đạt kết quả tương tự , vừa giúp chất lượng cây được duy trì.

=> Trong quá trình xóa phần tử khỏi R-tree, thuật toán sẽ tìm kiếm node chứa entry cần xóa và loại bỏ entry đó ra khỏi node đó. Sau đó, thuật toán sẽ kiểm tra xem node đó có bị underflow hay không. Nếu có, thuật toán sẽ chèn lại các phần tử đã xoá vào node đó hoặc node cha của nó cho đến khi không còn node nào bị underflow. Quá trình chèn lại các phần tử đã xoá này đã được mô tả trong thuật toán.

Tóm lại, thuật toán xóa phần tử khỏi R-tree đã xử lý underflow node bằng cách chèn lại các phần tử đã xoá vào cây. Việc chèn lại các phần tử này được ưu tiên hơn việc nối hai nút anh em lại với nhau như trong trường hợp của B+ -tree như các lý do đã trình bày bên trên.

### 1.3. Tổng kết:

Trong tất cả các biến thể cây R sẽ đề cập, việc duyệt cây (search) cho bất kỳ loại hoạt động nào được thực hiện theo cách chính xác như trong cây R nguyên thuỷ đã trình bày bên trên. Về cơ bản, các biến thể của cây R khác nhau ở cách chúng **thực hiện phân tách trong quá trình chèn** bằng cách xem xét các tiêu chí giảm thiểu khác nhau thay vì tổng diện tích của hai nút kết quả.

## 2. Các phiên bản động của R Tree:

Trong chương này, chúng tôi trình bày các phiên bản động của R-tree, trong đó các đối tượng được chèn vào trên cơ sở từng cái một - trái ngược với trường hợp có thể áp dụng một kỹ thuật đóng gói đặc biệt để chèn một đối tượng tĩnh đã biết trước - bằng cách tối ưu hóa chi phí lưu trữ và hiệu suất truy xuất. Trường hợp thứ hai sẽ được xem xét trong chương tiếp theo (Các phiên bản tĩnh của R Tree). Nói một cách đơn giản, ở đây chúng tôi tập trung vào cách thực hiện các **phép chèn và tách động** trong các biến thể cây R khác nhau.

### 2.1. The R+ -tree:

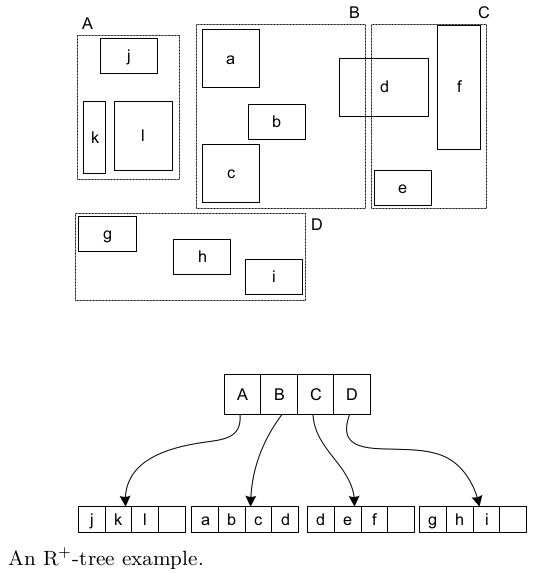
Cây R ban đầu có hai nhược điểm quan trọng thúc đẩy việc nghiên cứu các biến thể hiệu quả hơn:

- Việc thực hiện truy vấn vị trí điểm trong cây R có thể dẫn đến việc điều tra một số đường dẫn từ cấp độ gốc đến cấp độ lá. Đặc điểm này có thể dẫn đến suy giảm hiệu suất, đặc biệt khi có nhiều **sự chồng chéo của các MBR**.

- Một số hình chữ nhật lớn có thể làm tăng đáng kể mức độ chồng chéo, dẫn đến giảm hiệu suất trong quá trình thực hiện truy vấn phạm vi vì không gian trống.

=> R+-trees được đề xuất như một cấu trúc tránh truy cập nhiều đường dẫn trong các truy vấn vị trí điểm và do đó hiệu suất truy xuất có thể được cải thiện. Ngoài ra, tránh được sự chồng chéo của các MBR. Điều này đạt được bằng cách **sử dụng kỹ thuật cắt**. Nói một cách đơn giản, R+-trees không cho phép các MBR chồng lên nhau ở cùng cấp độ cây. Đổi lại, để đạt được điều này, các đối tượng được chèn phải được chia thành hai hoặc nhiều MBR hơn, điều đó có nghĩa là các entry của đối tượng cụ thể có thể được sao chép và lưu trữ dư thừa trong một số node.

Phía dưới là ví dụ về R+-trees, mặc dù cấu trúc trông tương tự như R-tree, lưu ý rằng đối tượng d được lưu trữ trong hai leaf node B và C. Ngoài ra, lưu ý rằng do cắt xén không tồn tại sự chồng lấp giữa các node tại cùng cấp độ (nên B, C không giao nhau và d bị lưu trữ thừa khi vừa lưu tại B, vừa lưu tại C).



Cách triển khai cấu trúc này và các giải thuật liên quan (cụ thể là thuật toán chèn và xóa) có thể xem ở class Java chúng tôi đã triển khai tại link Github sau:

### 2.2. The R\*-tree:

Như đã thảo luận, R-Tree chỉ dựa trên giảm thiểu diện tích mỗi MBR. Mặt khác, R\*-Tree vượt xa tiêu chí này và kiểm tra tiêu chí khác. Các tiêu chí R\*-Tree như sau:

- Giảm thiểu diện tích được bao phủ bởi mỗi MBR.

- Giảm thiểu sự chồng chéo giữa các MBR.

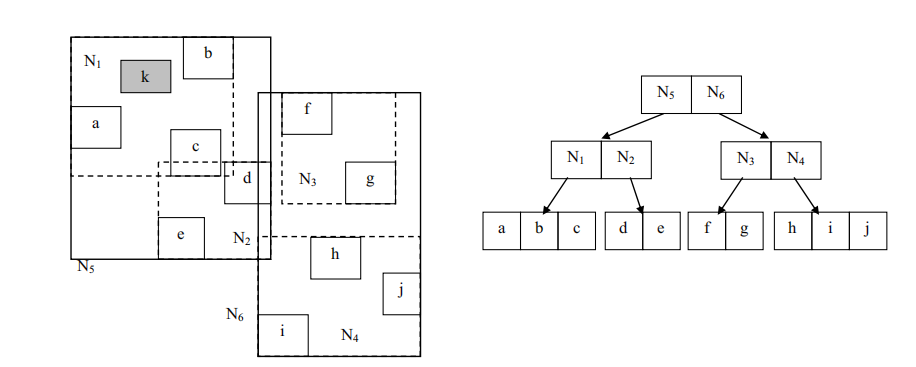
- Giảm thiểu chu vi MBR.

- Tối đa hóa việc sử dụng lưu trữ.

R\*-Tree tuân theo một phương pháp tìm kiếm sự kết hợp tốt nhất dựa trên các tiêu chí trên

Để chèn một entry mới, chúng ta phải quyết định nhánh nào sẽ đi theo, ở mỗi cấp độ của cây.

- Mô tả về R\*-Tree:



### 2.3. The Hilbert R-tree:

### 2.4. Linear Node Splitting:

### 2.5. Optimal Node Splitting:

### 2.6. Branch Grafting:

### 2.7. Compact R-trees:

### 2.8. cR-trees:

### 2.9. Deviating Variations:

#### 2.9.1. PR-trees:

#### 2.9.2. LR-trees:

### 2.10. Summary:

**R∗ -tree** tuân theo phương pháp kỹ thuật và đánh giá một số yếu tố ảnh hưởng đến hiệu suất của R-tree. Vì lý do này, nó được coi là biến thể mạnh mẽ nhất và đã tìm thấy nhiều ứng dụng, trong cả nghiên cứu hàn lâm và ứng dụng thương mại hoá. Tuy nhiên, nghiên cứu thực nghiệm trong đã chỉ ra rằng **Hilbert R-tree** có thể hoạt động tốt hơn các biến thể khác trong một số trường hợp. Điều đáng nói là **PR-trees**, mặc dù là một biến thể khác với các cây hiện có khác, nhưng là cách tiếp cận đầu tiên mang lại hiệu suất được đảm bảo trong **trường hợp xấu nhất** và khắc phục các **trường hợp suy biến** khi gần như toàn bộ cây phải được duyệt qua. Do đó, mặc dù thuật toán xây dựng phức tạp hơn, nó phải được coi là biến thể tốt nhất được báo cáo cho đến nay.

## 3. Các phiên bản tĩnh của R Tree:

### 3.1. The Packed R-tree:

### 3.2. The Hilbert Packed R-tree:

### 3.3. The STR R-tree:

### 3.4. Top-Down Packing Techniques:

### 3.5. Small-Tree-Large-Tree and GBI:

### 3.6. Bulk Insertion by Seeded Clustering:

### 3.7. The Buﬀer R-tree:

### 3.8. R-tree with Low Stabbing Number:

### 3.9. Merging R-trees:

### 3.10. Summary:

Công trình tiên phong của *The Packed R-tree* đã xác định (một năm sau khi R-tree ban đầu được đề xuất) lĩnh vực [bulk-loading algorithms](https://link.springer.com/article/10.1007/s00607-019-00709-4) cho các biến thể của R-tree.

Người ta cũng phải nhận ra việc sử dụng các đường cong cảm nhận không gian (ví dụ, trong *The Hilbert Packed R-tree*), như một cách tiếp cận đơn giản nhưng hiệu quả để [phân cụm các đối tượng không gian](https://warse.org/pdfs/2013/ijiti01232013.pdf).

Tuy nhiên, những cách tiếp cận này có thể gặp khó khăn khi [xử lý các không gian nhiều chiều](https://vi.wikipedia.org/wiki/Kh%C3%B4ng_gian_%C4%91a_chi%E1%BB%81u). *The STR R-tree* và *Top-Down Packing Techniques* cố gắng khắc phục một số thiếu sót này.

Cũng như trong trường hợp của các biến thể động, thật may mắn là các phương pháp đã được đề xuất mang lại hiệu suất được đảm bảo trong **trường hợp xấu nhất**. Đầu tiên, *R-tree with low stabbing number* (mục 3.8) giải quyết trường hợp xấu nhất bị ràng buộc đối với các **truy vấn điểm**.

Mặt khác, khi chuyển sang **cập nhật hàng loạt**, *STLT* (mục 3.5) đã xác định vấn đề này, trong khi *GBI* (mục 3.6) cố gắng khắc phục **sự phụ thuộc của *STLT* đối với dữ liệu sai lệch**. *Merging R-trees* (mục 3.9) có thể được coi là một giải pháp tao nhã và trên thực tế, nó là một trong những cách tiếp cận tốt nhất cho vấn đề **cập nhật hàng loạt**. Cuối cùng, thật thú vị khi nhận thấy vấn đề hợp nhất hai cây R hiện có, đây không thể được coi là trường hợp đặc biệt của cập nhật hàng loạt, bởi vì cả hai cây R đều có cùng kích thước. Như vậy, *Merging R-trees* đáng được đề cập cho ứng dụng này.

# Part II. Vấn đề xử lý truy vấn.

## 4. Kỹ thuật xử lý truy vấn cơ bản:

- Các kỹ thuật cơ bản để xử lý truy vấn không gian bằng R-Tree:

* Truy vấn phạm vi và tôpô (tìm dữ liệu thỏa mãn vùng truy vấn, cùng với quan hệ tôpô của chúng đối với vùng này).
* Truy vấn lân cận gần nhất (tìm dữ liệu gần điểm truy vấn nhất).
* Truy vấn nối không gian (tìm các cặp dữ liệu thỏa mãn một điều kiện).

### 4.1. Xử lý hai bước:

- Xử lý các truy vấn không gian đưa ra các yêu cầu quan trọng vì phải xử lý khối lượng dữ liệu lớn và sự phức tạp của cả đối tượng và truy vấn.

- Để có hiệu quả truy vấn không gian phải tận dụng được khoảng cách gần của các đối tượng bằng R-Tree để tập trung tìm các đối tượng đáp ứng truy vấn.

- Lý do chính cho sự phổ biến của các chỉ số R-Tree bắt nguồn từ tính linh hoạt của chúng, bởi chúng có thể hỗ trợ nhiều loại toán tử không gian:

* Các toán tử tôpô: tìm tất cả các đối tượng

VD: trùng lặp hoặc bao phủ một đối tượng nhất định

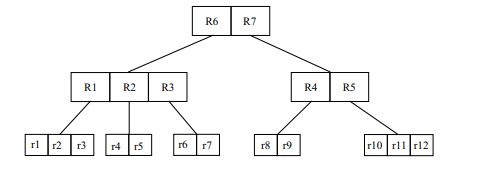
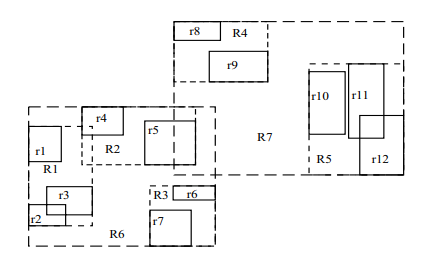
* Toán tử định hướng: tìm tất cả các đối tượng

VD: nằm ở phía Bắc của một đối tượng nhất định

* Toán tử khoảng cách: tìm tất cả các đối tượng

VD: nằm nhỏ hơn một khoảng cách nhất định từ một đối tượng nhất định 76(truy vấn phạm vi) hoặc tìm k các đối tượng gần nhất với một đối tượng nhất định (truy vấn K Nearest - Neighbor).

Cụ thể: Chẳng hạn, để một tập hợp gồm 10 hình chữ nhật 2 chiều, được kí hiệu là r1,...,r10 (hình bên trên) và được tổ chức với R-Tree (hình bên dưới). Đưa ra một điểm truy vấn a, câu trả lời cho truy vấn tìm tất cả các đối tượng bao phủ a (toán tử tô pô) là r11 và r12. Với điểm truy vấn b, câu trả lời cho truy vấn tìm tất cả các đối tượng ở phía bắc (toán tử định hướng) là r8. Tập trung vào truy vấn Nearest-Neighbor (toán tử khoảng cách), với điểm truy vấn c, r5 sẽ được tìm thấy. Cuối cùng, xem xét truy vấn phạm vi, đặt truy vấn điểm d và khoảng cách là bán kính của vòng tròn được mô tả bằng đường nét đứt, ta sẽ tìm thấy r9 và r10.

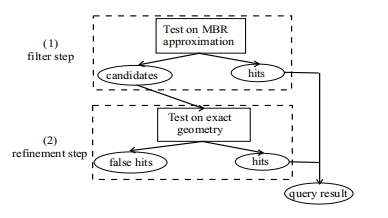


- R-Tree trừu tượng hóa các đối tượng có hình dạng phức tạp bằng cách sử dụng các xấp xỉ MBR (Minimum Bounding Rectangle) của chúng. Để trả lời các truy vấn có chứa các toán tử , quy trình gồm hai bước:

* Lọc: tập hợp tất cả các đối tượng có MBR thỏa mãn truy vấn đã cho được tìm thấy, bao gồm candidate set dù nó có MBR đáp ứng được điều kiện của truy vấn hay không.
* Sàng lọc: hình học thức tế của từng member của candidate set được kiểm tra để loại bỏ các lần truy cập sai và tìm câu trả lời cho truy vấn.

\* Note: bước lọc không thể xác định việc đưa một đối tượng vào kết quả truy vấn. Tuy nhiên, có một toán tử (hầu hết là toán tử định hướng) cho phép tìm kết quả truy vấn từ bước lọc.

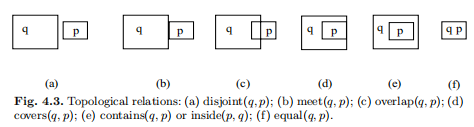
\* Mô tả quy trình xử lý truy vấn hai bước:



### 4.2. Truy vấn phạm vi và truy vấn tô pô:

- Hoạt động phổ biến nhất với R-Tree là truy vấn phạm vi, truy vấn này tìm tất cả các đối tượng mà một vùng truy vấn giao nhau.

- Việc truy vấn phạm vi được mô tả như sau:



- Các đối tượng được lọc bằng cách kiểm tra xem MBR của chúng có giao với vùng truy vấn hay không. Điểm truy vấn có thể được coi là một truy vấn phạm vi và toán tử giao nhau có thể được coi là một trường hợp đặc biệt của các quan hệ tôpô.

- Các mối quan hệ tôpô có thể được sử dụng để giảm thiểu chi phí I/O cho các truy vấn liên quan đến quan hệ tôpô. Cụ thể, việc phân tích, lưu trữ và truy xuất dữ liệu không gian có thể tốn nhiều tài nguyên hệ thống. Bằng cách sử dụng các mối quan hệ tô pô người dùng có thể giảm bớt dữ liệu cần được xử lý hoặc truy vấn, giúp giảm thiểu việc I/O dữ liệu vào bộ nhớ và giảm độ phức tạp của truy vấn.

- Mối quan hệ tô pô giữa hai MBR không nhất thiết trùng với quan hệ tô pô giữa hai đối tượng tương ứng. Ví dụ: hai MBR có thể chứa hoặc giao nhau, nhưng các đối tượng bên trong chúng lại không chứa hoặc giao nhau.

### 4.3. Truy vấn Nearest-Neighbor:

- Truy vấn này mô tả thủ tục tìm kiếm nhanh và giới hạn để tránh việc kiểm tra toàn bộ nội dung chỉ mục.

- Phương pháp này dựa trên các độ đo khoảng cách được phát triển, đo lường khoảng cách lạc quan và bi quan giữa các nội dung R-tree và điểm truy vấn.

* Khoảng cách lạc quan được tính toán dựa trên kỳ vọng rằng các nội dung trong R-Tree có thể gần với điểm truy vấn.
* Khoảng cách bi quan được tính toán dựa trên giá định rằng các nội dung trong R-Tree có thể xa khỏi điểm truy vấn.

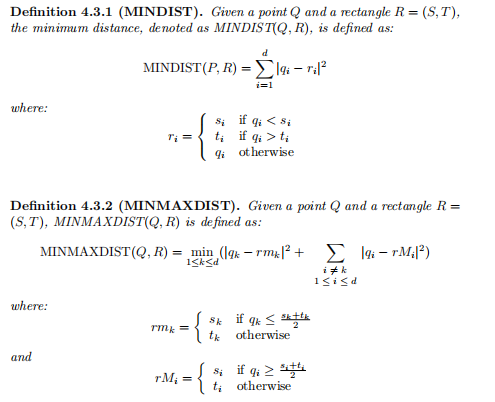
=> Phương pháp này giúp giảm thiểu số lượng nội dung cần phải kiểm tra trong quá trình truy vấn, giúp gia tăng tốc độ xử lý và giảm bớt tải cho hệ thống.

- Đối với tìm kiếm Nearest-Neighbor, có hai độ đo là MINDIST và MIMAX DIST được sử dụng để đưa ra giới hạn dưới và giới hạn trên cho khoảng cách thực sự giữa B-Tree và điểm truy vấn Q.

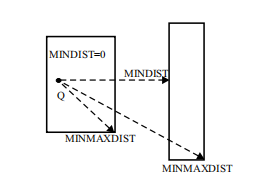
* MINDIST (Minimum Distance): đây là độ đo khoảng cách lạc quan, MINDIST được sử dụng làm giới hạn dưới cho khoảng cách thực sự, tức là nếu khoảng cách giữa điểm truy vấn Q và giói hạn trên của nội dung B-Tree (giới hạn trên ban đầu được đặt là vô cực) nhỏ hơn MINDIST thì không cần kiểm tra nội dung này và có thể bỏ qua trong truy vấn.
* MINMAXDIST (Maximum Distance): đây là độ đo khoảng cách bi quan, MINMAXDIST được sử dụng làm giới hạn trên cho khoảng cách thực sự, tức là khoảng cách giữa điểm truy vấn Q và giới hạn dưới của nội dung B-Tree (giới hạn dưới ban đầu được đặt bằng 0) lớn hơn MINMAXDIST thì không cần kiểm tra nội dung B-Tree này và có thể bỏ qua trong truy vấn.

=> Sử dụng MINDIST và MINMAXDIST giúp giúp giới hạn phạm vi tìm kiếm trong B-Tree, giúp tăng tốc độ tìm kiểm và giảm bớt số lượng nội dung B-Tree cần phải kiểm tra.

- Khoảng cách được định nghĩa trong không gian Euclide đa chiều, với mỗi hình chữ nhật được biểu diễn bằng hai đầu mút của đường chéo chính của nó.



- Mô tả về MINDIST và MIMAX DIST:



#### 4.3.1. Thuật toán rẽ nhánh và giới hạn:

- Thuật toán duyệt branch-and-bound sử dụng các độ đo trên để sắp xếp và cắt bớt cây tìm kiếm.

- Thứ tự duyệt cây được xác định bởi phương pháp sắp xếp các node duyệt trong quá trình duyệt cây.

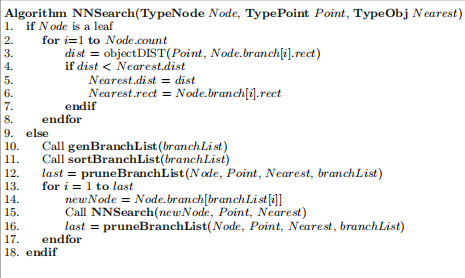
* Khi sử dụng độ đo MINDIST, cây sẽ được duyệt theo thứ tự các node gần đến xa hơn so với điểm truy vấn Q.
* Khi sử dụng độ đo MINMAXDIST, cây sẽ được duyệt theo thứ tự các node xa đến gần điểm truy vấn Q.

- Độ đo MINDIST cho ra thứ tự tối ưu hơn so với độ đo MIMAX DIST, nhưng cũng có trường hợp các tập dữ liệu thì độ đo MIMAXDIST cho ra kết quả tìm kiếm ít tốn kém hơn.

- Để cắt tỉa các node tìm kiếm ta dựa trên hướng dẫn (heuristics):

* H1: Cắt tỉa từ trên xuống (downward pruning) bằng cách so sánh khoảng cách nhỏ nhất từ điểm truy vấn Q đến hình chữ nhật bao quanh MBR M với khoảng cách MINMAXDIST(Q,M’) của MBR M’ khác M’. Nếu khoảng cách từ Q đến M lớn hơn khoảng cách MINMAXDIST(Q,M’) của M’ thì M không thể chứa đối tượng NN và có thể bị loại bỏ.
* H2: Cắt tỉa từ trên xuống (downward pruning) bằng cách so sánh khoảng cách thực tế từ Q đến đối tượng O với khoảng cách MINMAXDIST(Q,M) của MBR M. Nếu khoảng cách thực tế từ Q đến O lớn hơn MINMAXDIST(Q,M) thì M không thể chứa đối tượng O gần nhất và có thể bị loại bỏ.
* H3: Mỗi MBR M có khoảng cách nhỏ nhất từ Q đến M lớn hơn khoảng cách thực tế từ Q đến đối tượng O thì M không thể chứa đối tượng nào gần hơn O và có thể bị loại bỏ. Đây là cách cắt tỉa từ dưới lên (upward pruning).

- Các bước thực hiện thuật toán:

* Thuật toán bắt đầu từ node gốc và khởi tạo biến Nearest là vô cùng.
* Ở mỗi node thuật toán tìm khoảng cách nhỏ nhất (MINDIST) cho các MBR con của node (bước 10) và sắp xếp chúng, cùng với con trỏ tương ứng đến các node con vào danh sách nhánh hoạt động (bước 11).
* Sau đó, cắt tỉa theo loại 1 và loại 2 được thực hiện (bước 12).
* Đối với mỗi thành viên của danh sách nhánh hoạt động, thuật toán được áp dụng đệ quy (bước 15).
* Ở mức lá (bước 1-8), hàm khoảng cách thực tế được sử dụng (bước 3) và biến Nearest được cập nhật tương ứng (bước 4-7).
* Cuối cùng, cắt tỉa theo loại 3 được áp dụng (bước 16).

#### 

#### 

#### 

#### 

#### 

#### 

#### 

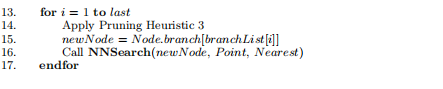
#### 4.3.2. Cải thiện thuật toán gốc:

- Thuật toán branch-and-bound có thể được phát triển khi xem xét việc cắt tỉa dựa trên số lần truy cập vào các node.

- Phương pháp của họ dựa trên quan sát rằng chỉ cần sử dụng cắt tỉa H3 để duy trì cùng số lượng node bị cắt tỉa (tức là không truy cập).

- Tuy nhiên, cắt tỉa H3 này phải được áp dụng ở vị trí khác, dẫn đến một thuật toán tìm kiếm Nearest-Neighbor được sửa đổi.

- Vì vậy, thủ tục Search được sửa đổi bằng cách loại bỏ bước 12 (áp dụng cắt tỉa H1 và H2) và đặt lại bước 3 (cắt tỉa H3) trước khi áp dụng đệ quy của bước 15. Do đó, vòng lặp của các bước 13-17 được thực hiện.

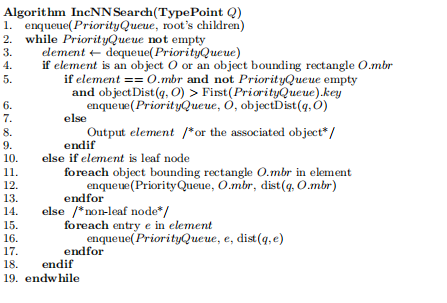


#### 4.3.3. Tim kiếm Nearest-Neighbor gia tăng:

- Thuật toán này tìm các đối tượng dữ liệu theo thứ tự khoảng cách từ đối tượng truy vấn (theo thứ hạng).

- Phương pháp này khác với việc tìm kiếm k hàng xóm gần nhất được định trước, vì giá trị k không thể biết trước.

- Thuật toán duy trì tập hợp các node cần được thăm qua một hàng đợi ưu tiên và được sắp xếp theo độ đo khoảng cách MINDIST.



#### 4.3.4. So sánh các thuật toán Nearest-Neighbor:

- Có 3 điều ảnh hưởng đến hiệu suất thuật toán:

* Tiêu chí sắp xếp thứ tự các node được truy cập
* Cách thức sắp xếp thứ tự các node được truy cập
* Heuristics để loại bỏ nhánh không cần thiết (pruning)

- Kết luận:

* INN vượt trội hơn KNN trong các hoạt động tìm kiếm điểm gần nhất.
* Với tìm kiếm k điểm gần nhất (k được xác định trước), có sự cải thiện 20%-30% về thời gian thực thi so với KNN.
* INN cũng cho thấy khả năng mở rộng tốt đến các bộ dữ liệu lớn, có tốc độ tăng từ 1.8 đến 5.8 lần so với KNN.
* Dù INN là một phương pháp tối ưu của KNN nhưng cũng cần xem xét đến độ phức tạp tính toán và khả năng triển khai trong thực tế.

### 4.4. Truy vấn nối không gian:

- Trong trường hợp có hai tập hợp dữ liệu không gian A = {a1, ..., an} và B = {b1, ..., bm} (trong đó ai, bi là các đối tượng không gian), nối không gian là quá trình tính toán tất cả các cặp (ai, bj), trong đó ai thuộc A và bj thuộc B, thỏa mãn một mệnh đề không gian.

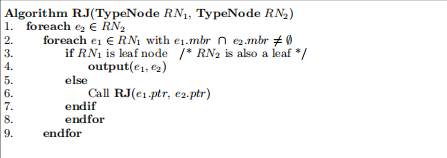
- Loại truy vấn này đặt yêu cầu cao hơn về hiệu suất xử lý truy vấn hiệu quả.

#### 4.4.1. Thuật toán dựa trên duyệt theo độ sâu:

- Thuật toán này duyệt qua hai R-Tree theo chiều sâu, kiểm tra các mục nhập của hai node từ hai cây.

- Nếu MBR (Minimum Bounding Rectangle) của chúng không giao nhau, thì không cần phải kiểm tra tiếp các cây con tương ứng. Nếu không, thuật toán tiếp tục đệ quy đến các mục nhập của các cây con.

- Mô tả thuật toán:



- Thuật toán này có thể được cải tiến bằng cách sử dụng các kỹ thuật như:

* Giảm bớt thời gian CPU: Kiểm tra sơ bộ (Preliminary Checking), Sắp xếp các MBR trước khi so sánh
* Điều chỉnh thời gian I/O: Sử dụng kĩ thuật đệm (Buffering), Sử dụng kĩ thuật đọc trước (Prefetching)
* Một số cải tiến khác: Sử dụng kĩ thuật tiền xử lý (Preprocessing), Sử dụng các biểu diễn dữ liệu gọn nhẹ (Lightweight Data Representation), Tối ưu hóa cấu trúc cây R-Tree

#### 4.4.2. Thuật toán dựa trên duyệt theo chiều rộng:

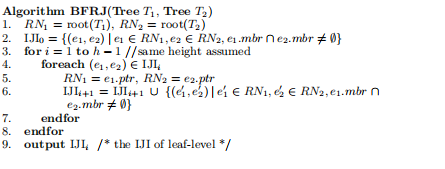
- Thuật toán này sử dụng duyệt qua các mức của cả hai cây R-Tree theo chiều rộng.

- Thuật toán này sử dụng các tối ưu hóa toàn cục để cải thiện việc đọc/ghi dữ liệu.

- BFRI (Breadth-First R-Tree Join) đưa ra ba tối ưu hóa mới cho việc quản lý bộ nhớ đệm, thứ tự của kết quả và lưu trữ nội dung tạm thời trên đĩa.

- Kết luận: BFRI cho thấy hiệu suất tốt hơn so với thuật toán RJ (R-Tree Join).

- Mô tả thuật toán:



- Bảng so sánh giữa hai thuật toán:

#### 

#### 4.4.3. Nối giữa tập dữ liệu được lập chỉ mục trên R-Tree và không được lập chỉ mục trên R-Tree:

- Trong trường hợp kết quả truy vấn trung gian (ví dụ: của truy vấn phạm vi) tham gia vào liên kết, cây R sẽ không khả dụng cho kết quả đó. Một cách tiếp cận đơn giản để thực hiện phép nối trong trường hợp này là áp dụng nhiều truy vấn phạm vi, mỗi truy vấn cho từng đối tượng trong tập dữ liệu không được lập chỉ mục, trên cây R của tập dữ liệu khác. Rõ ràng, cách tiếp cận này chỉ hiệu quả khi kích thước của tập dữ liệu trung gian rất nhỏ.

- Một cây R có thể được tạo (ví dụ: bằng cách nạp tải tập dữ liệu hàng loạt) cho tập dữ liệu không được lập chỉ mục và áp dụng các thuật toán đã được mô tả để nối hai R-Tree.

- Có thể sử dụng cây R-tree hiện có như một bộ xương (skeleton) để xây dựng cây con cho tập dữ liệu không được lập chỉ mục.

- Có thể sắp xếp và so khớp các đối tượng của tập dữ liệu không được lập chỉ mục (sử dụng thứ tự không gian), tạo các nút lá có thể chứa trong bộ nhớ chính, và kiểm tra từng nút lá này với các lá của cây R-tree hiện có của tập dữ liệu khác đối với điều kiện nối.

- Có thuật toán ưu tiên hàng đợi (Priority Queue-Driven Traversal - PQ) kết hợp cả hai phương pháp dựa trên chỉ mục và không dựa trên chỉ mục.

- Có phương pháp gọi là spatial join dựa trên chỉ số khe (slot index spatial join - SISJ), sử dụng cấu trúc của cây R-tree hiện có để xác định phạm vi của các phân vùng không gian.

- Có phương pháp kết hợp giữa Quadtree và R-tree để thực hiện phép nối giữa các cấu trúc dữ liệu này.

- Có một số phương pháp đặc biệt khác để xử lý tập dữ liệu không được lập chỉ mục, nhưng không có liên quan trực tiếp đến việc xử lý truy vấn cho các cây R-tree hiện có.

4.5. Tóm tắt:

- Phạm vi và tìm kiếm lân cận gần nhất là các hoạt động quan trọng trong cấu trúc dữ liệu và R-Tree đã được sử dụng rộng rãi cho các hoạt động này.

- Thuật toán tìm kiếm phạm vi của cây R gốc được sử dụng trong hầu hết các biến thể nhưng có thể phát triển các thuật toán hiệu quả hơn bằng cách thay đổi đáng kể cấu trúc cây gốc.

- Về các thuật toán lân cận gần nhất, như đã mô tả, một số các phương pháp đã được đề xuất. Thuật toán được đề xuất bởi Hjaltason và Samet là tối ưu về số node được truy cập. Khả năng “duyệt” của nó cũng rất hấp dẫn, bởi vì mô hình truy vấn này phù hợp với nhiều ứng dụng. Tuy nhiên, đây vẫn là một vấn đề mở để xác định một thuật toán tối ưu liên quan đến chi phí tổng thể, cũng như chi phí để thao tác với các cấu trúc dữ liệu phụ trợ (chẳng hạn như hàng đợi ưu tiên).

- Tầm quan trọng của hoạt động nối không gian dẫn đến sự tồn tại của nhiều thuật toán nối, đồng thời xem xét một số yếu tố (ví dụ: kiểu duyệt qua, số lượng cây đã được kết nối, số lượng bộ dữ liệu mà cây R đã được xây dựng, v.v.). Nghiên cứu về các thuật toán nối và Nearest-Neighbor đã ảnh hưởng đến các loại truy vấn phức tạp hơn khác, bởi vì truy vấn NN và truy vấn nối không gian có thể được coi là nguyên thủy cho các hoạt động phức tạp hơn (ví dụ: truy vấn khai thác dữ liệu).

## 5. Xử lý các truy vấn phức tạp hơn:

- Trước hết chúng tôi mô tả các phần mở rộng cho các truy vấn phạm vi, tiếp theo chúng tôi xem xét các phần mở rộng cho các truy vấn lân cận gần nhất và cuối cùng chúng tôi trình bày sự mở rộng cho các truy vấn nối không gian.

- Các truy vấn này yêu cầu chi phí CPU và I/O đáng kể, do đó các phương pháp tinh vi đã được áp dụng để giảm chi phí xử lý. Cuối cùng, các kỹ thuật xử lý truy vấn gần đúng được trình bày nhằm hy sinh độ chính xác để có hiệu suất tốt hơn.

### 5.1. Truy vấn phạm vi phân loại:

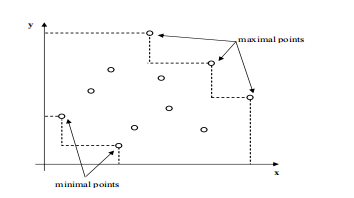
- Trong một số lượng lớn các ứng dụng, các đối tượng có thể được tổng hợp thành các nhóm (rời rạc). Một biến thể được hình thành là tìm kiếm các nhóm (thay vì chính các đối tượng riêng lẻ) được giao nhau bởi vùng truy vấn. Loại truy vấn này được ký hiệu là Categorical Range Query (CRQ).

- CRQ có thể được coi là một truy vấn phạm vi đặc biệt. Như vậy, nó có thể được xử lý bằng cách thực hiện truy vấn phạm vi thông thường trước tiên, sau đó lọc bộ đầu ra của nó để chọn các danh mục riêng biệt.

- Trong cơ sở dữ liệu không gian, các đối tượng gần nhau trong không gian có thể chia sẻ cùng một giá trị cho một thuộc tính phân loại, do đó, độ chọn lọc của CRQ có thể lớn hơn nhiều (nghĩa là kích thước đầu ra nhỏ hơn) so với truy vấn phạm vi thông thường. Trong trường hợp như vậy, bằng cách xử lý truy vấn phạm vi đơn giản trước tiên và sau đó lọc theo thuộc tính phân loại, chi phí sẽ cao.

- Dựa trên R-trees, phát triển một chỉ số nhiều cây tích hợp một cách hiệu quả các kích thước không gian và thuộc tính phân loại. Cách tiếp cận này dựa trên khái niệm rộng về việc tăng cường các nút với thông tin bổ sung để tăng tốc các truy vấn. Sơ đồ tăng cường dựa trên khái niệm về điểm tối đa/tối thiểu.

- Mô tả về tập hợp các điểm hai chiều và các điểm cực đại/cực tiểu tương ứng:



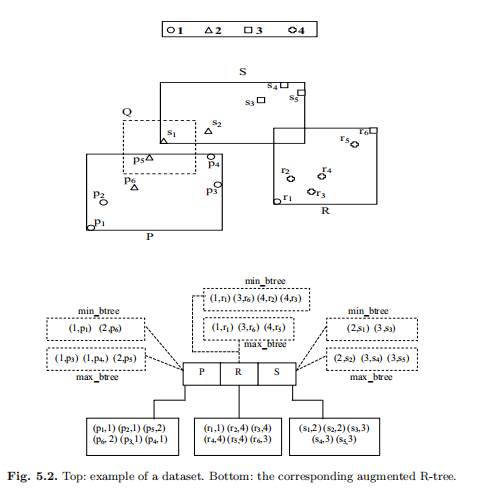
- Mỗi danh mục trong cây con của cây R được biểu diễn riêng biệt thông qua các điểm cực đại/tối thiểu của nó và tập hợp các điểm như vậy cho tất cả các danh mục trong cây con bao gồm thông tin làm cơ sở cho việc tăng thêm.

- Cụ thể, cây R được tăng cường với các cấu trúc dữ liệu phụ trợ được duy trì tại mỗi mục nhập nút bên trong (nghĩa là không phải lá). Mỗi mục nhập e trong một nút R-Tree bên trong có dạng: (e.mbr, e.ptr, e.max\_btree, e.min\_btree , trong đó

e.max\_btree và e.min\_btree là các con trỏ tới gốc của B+ - cây lưu trữ các điểm cực đại và cực tiểu tương ứng trong cây con của e (hai phần tử còn lại, e.mbr và e.ptr, có các giá trị chính quy của chúng nghĩa).

- Trong phạm vi B+ - cây, các điểm tối đa và tối thiểu được sắp xếp theo các giá trị danh mục của chúng, bao gồm các khóa tìm kiếm trong mỗi B+ - tree. Do đó, hai cây B+ của mục nhập nút e được thăm dò đối với các giá trị loại và có thể truy xuất các điểm cực đại và cực tiểu.

- Phần trên cùng của Hình 5.2 minh họa một tập dữ liệu mẫu (mô tả các danh mục cũng được mô tả) và phần dưới cùng hiển thị chỉ mục nhiều cây tương ứng. Lưu ý rằng các nút R-tree được mô tả bằng các đường liền nét trong khi những cái B+ -tree được vẽ bằng các đường đứt nét. Các điểm trong các lá của cây R được lưu trữ cùng với danh mục của chúng, nhưng không có thứ tự đối với các giá trị danh mục. Ngược lại, trong các nút B+ -tree, các mục nhập là các điểm tối đa/tối thiểu và chúng được sắp xếp theo giá trị danh mục của chúng. Chẳng hạn, tại nút P, điểm tối đa cho loại 1 là điểm p3 và p4, và cho loại 2 là điểm p5. Đối với cùng một nút, các điểm tối thiểu là: p1 cho loại 1 và p6 cho loại 2.



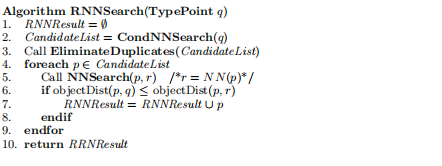
- Do các thuộc tính của các điểm cực đại và cực tiểu, các nhánh của cây R có thể bị cắt bớt trong quá trình kiểm tra CRQ. Ví dụ, xem xét hình chữ nhật truy vấn Q được mô tả ở phần trên cùng của Hình 5.2 và cây R ở phần dưới cùng của Hình 5.2. Bắt đầu từ nút có MBR là hình chữ nhật P, nội dung của btree cực đại của nó (cây B có các điểm cực đại) và btree cực tiểu (cây B có các điểm cực tiểu) được kiểm tra. Điểm p5 được chứa trong Q, do đó loại 2 được bao gồm trong tập hợp đầu ra. Không tìm thấy điểm nào cho loại 3 và 4, trong khi chúng ta không thể xác định từ các điểm p1, p3 và p4 liệu loại 1 có tồn tại trong Q hay không. Do đó, chúng tôi chuyển sang lá con của P, nơi không tìm thấy điểm nào từ loại 1 là chứa trong Q. Quá trình tìm kiếm tiếp tục tại các nút bên trong, kiểm tra mục nhập có MBR là hình chữ nhật S; mục nhập có MBR R bị bỏ qua vì R không bị giao bởi Q. Trong nút này, chúng tôi đang tìm kiếm các loại 1, 3 và 4 vì 2 đã được xuất. Không có điểm tối đa hoặc tối thiểu cho loại 1 và 4, trong khi đối với loại 3 chúng ta có thể xác định rằng nó không tồn tại trong Q (vì điểm cực tiểu s3 chiếm cực đại ở góc trên bên phải của Q). Do đó, chúng ta không cần phải đi xuống cây con của nút này và quá trình tìm kiếm kết thúc khi đã bao gồm loại 2 trong tập hợp đầu ra. So sánh thực nghiệm với dữ liệu thực và dữ liệu tổng hợp trong [153] minh họa tính ưu việt của phương pháp được đề xuất so với truy vấn phạm vi đơn giản và sơ đồ ngây thơ để tăng cường cây R. Tùy thuộc vào giá trị của độ chọn lọc tương đối của CRQ, cách tiếp cận đã nói ở trên có thể đạt được những cải tiến về hiệu suất lên đến một mức độ lớn, đối với các bộ dữ liệu thực lớn.

### 5.2. Truy vấn Nearest-Neighbor bị ràng buộc và đảo ngược:

#### 5.2.1. Nearest-Neighbors đảo ngược:

- Truy vấn hàng xóm gần nhất (RNN) đảo ngược tìm tập hợp các điểm cơ sở dữ liệu có điểm truy vấn là hàng xóm gần nhất. Stanoi et al. đã phát triển một thuật toán đảo ngược hàng xóm gần nhất cho cây R, thuật toán này có thể xử lý dữ liệu động một cách hiệu quả.

- Mô tả thuật toán:

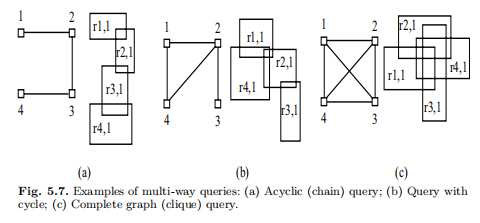


#### 5.2.2. Tìm kiếm Nearest-Neighbor bị ràng buộc tổng quát:

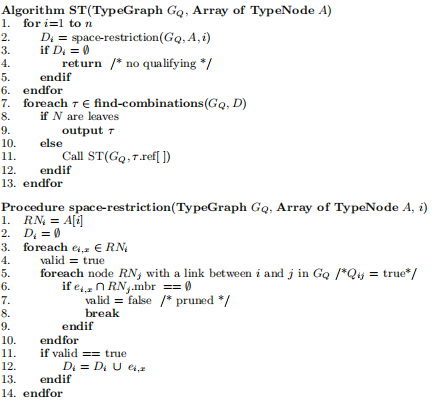
- Nghiên cứu của Ferhatosmanoglou và đồng nghiệp mở rộng khái niệm tìm kiếm gần nhất với điều kiện (CNN) bằng cách xem xét các ràng buộc tổng quát hơn, chẳng hạn như đa giác lồi, để chỉ định vùng quan tâm. Trong phương pháp này, một truy vấn CNN tìm kiếm thành phố gần nhất thỏa mãn cả ràng buộc về khoảng cách và hướng. Ví dụ, với một bản đồ 2 chiều, truy vấn CNN sẽ tìm thành phố gần nhất nằm về phía nam của điểm truy vấn. Nghiên cứu này đề xuất một phương pháp mới kết hợp cả hai điều kiện gần nhất và ràng buộc vùng trong một giai đoạn. Phương pháp này dựa trên một sự mở rộng của công việc đã được mô tả cho bài toán tìm kiếm hàng xóm gần nhất ngược lại, trong đó xem xét các vùng được xác định bởi đa giác thay vì các đường chia không gian. Các chỉ số MINDIST và MIMAX DIST cũng được điều chỉnh theo cách khác trong phương pháp này.

### 5.3. Truy vấn nối không gian đa chiều:

- Ví dụ:



- Mô tả thuật toán:



# 

# 

# 

# 

# 

# 

# Part III. Ứng dụng của R-trees

## 6. R-trees trong cơ sở dữ liệu không gian:

Hệ cơ sở dữ liệu không thời gian nhằm kết hợp các đặc tính không gian và thời gian của dữ liệu. Có nhiều ứng dụng từ việc xử lý hiệu quả các truy vấn không gian thời gian, chẳng hạn như: hệ thống liên lạc di động, hệ thống kiểm soát giao thông (VD: giám sát giao thông hàng không), hệ thống thông tin địa lý (GIS), đa phương tiện và các dịch vụ dựa trên vị trí (LBS). Ta cần các kỹ thuật và công cụ để tăng khả năng xử lý cả hai đặc tính không gian và thời gian một cách hiệu quả.

### 6.1. Mở đầu:

Các phương pháp truy cập không gian thời gian (STAMs) cung cấp các công cụ cần thiết phải để truy vấn dữ liệu không gian thời gian. Dữ liệu không gian thời gian được đặc trưng bởi sự thay đổi về vị trí hoặc hình dạng theo thời gian. Nhiều phương pháp được đề xuất dựa trên cấu trúc R-tree đã được biết đến.

Các phương pháp truy vấn và xử lý truy vấn cho cơ sở dữ liệu không gian thời gian được chia thành các lĩnh vực sau:

* Kỹ thuật xử lý truy vấn cho vị trí quá khứ của đối tượng
* Kỹ thuật xử lý truy vấn cho vị trí hiện tại và tương lai của đối tượng.

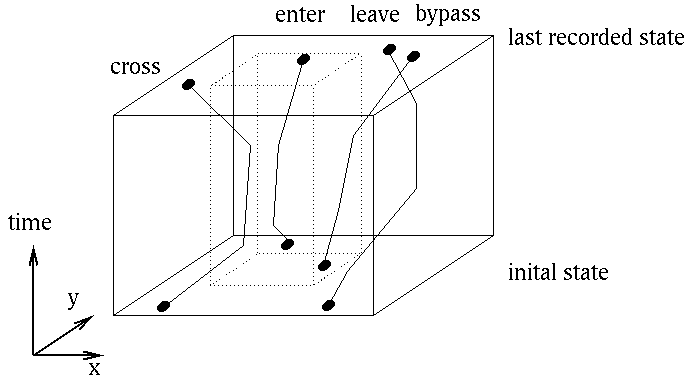
Việc thêm thời gian như một tham số truy vấn làm tăng số lượng truy vấn mà người dùng có thể thực hiện. Các truy vấn tập trung vào một điểm thời gian cụ thể được gọi là truy vấn time-slice, trong khi những truy vấn tập trung vào khoảng thời gian được gọi là truy vấn time interval. Khi kết hợp với các tiền đề không gian và khả năng truy vấn quá khứ, hiện tại, tương lai, các truy vấn không thời gian có thể trở nên rất phức tạp và đòi hỏi nỗ lực xử lý đáng kể.

Giả sử chúng ta duy trì một cơ sở dữ liệu về các máy bay, và chúng ta ghi lại vị trí của mỗi máy bay và thời điểm tương ứng. Chúng tôi trình bày một danh sách các truy vấn ví dụ có thể được đưa ra cho một cơ sở dữ liệu như vậy:

* Những máy bay nào đã bay qua vùng ba chiều R tại thời điểm $$t\_{x}$$?
* Những máy bay nào đã bay qua vùng ba chiều R trong khoảng thời gian $$\left[ t\_{start}, t\_{end} \right]$$?
* k máy bay gần nhất với máy bay a tại thời điểm $$t\_{x}$$ là gì?
* Cho các đặc tính chuyển động hiện tại của các máy bay, dự đoán máy bay gần với máy bay a tại một thời điểm $$t\_{x}$$ trong tương lai gần.
* Những máy bay nào sẽ bay qua vùng ba chiều R trong khoảng thời gian $$\left[ t\_{start}, t\_{end} \right]$$?
* Xác định các cặp máy bay $$\left( a\_{i}, a\_{j} \right)$$ sao cho khoảng cách của chúng tại thời điểm $$t\_{x}$$ nhỏ hơn 5 Km.

Qua các truy vấn đề cập ở trên, rõ ràng với các truy vấn cơ bản như phạm vi (range), hàng xóm gần nhất (nearest neighbors), kết hợp (join), cặp gần nhất (closet pairs)… có thể kết hợp với truy vấn time-slice hoặc time interval cho quá khứ, hiện tại hoặc tương lai để tạo thành các truy vấn phức tạp hơn.

Các truy vấn không gian thời gian topo bao gồm quỹ đạo đối tượng và bao gồm cả ý nghĩa như lối vào và lối đi, không thể phân rã thành các truy vấn dựa trên tọa độ. Những truy vấn này quan trọng trong các ứng dụng như hệ thống quản lý đội xe nhưng lại tốn kém trong quá trình xử lý. Hỗ trợ chỉ mục cho truy vấn topo không gian thời gian không phải là điều dễ dàng vì khái niệm quỹ đạo phải được bảo tồn trong chỉ mục.

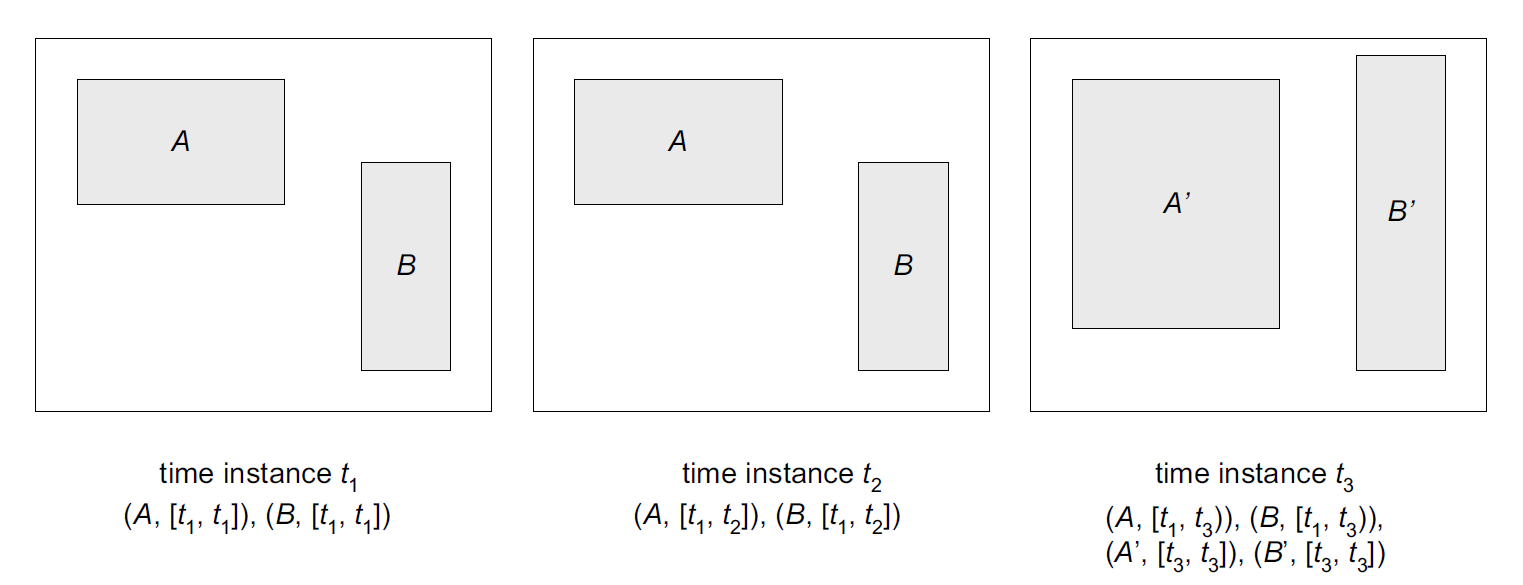


*Truy vấn không gian thời gian topo*

Dưới đây là một số phương pháp STAMs và các kỹ thuật xử lý truy vấn liên quan đến các ứng dụng không gian hai điểm, giám sát quỹ đạo và đối tượng chuyển động, Tất cả các phương pháp sau đều dưa trên khái niệm R-tree.

### 6.2. RT-tree:

Đây là phương pháp đơn giản nhất để tích hợp thời gian vào phương pháp truy cập R-tree. Về cơ bản, RT-tree là một R-tree được bổ sung thêm thông tin về thời gian trong mỗi node. Tuy nhiên, quá trình xây dựng cây được quản lý bởi thông tin không gian vì thế việc xử lý các truy vấn thời gian gây khó khăn trong việc tính toán. Mỗi đối tượng trong RT-Tree có một phạm vi không gian và thời gian. Nếu đối tượng thay đổi phạm vi không gian, một mục mới được tạo ra và phạm vi thời gian được khởi tạo lại từ thời điểm mới đó. Nếu đối tượng thay đổi phạm vi không gian thường xuyên, một số lượng lớn các mục sẽ được tạo ra và tiêu thụ một lượng lớn bộ nhớ. Hình dưới đây minh họa một ví dụ, cho thấy tập hợp các mục được xử lý bởi RT-Tree cho ba thời điểm khác nhau. Ở thời điểm thứ ba, cả hai đối tượng thay đổi phạm vi không gian, dẫn đến hai mục mới được thêm vào RT-Tree.



*Các mục được xử lý bởi RT-tree trong ba trường hợp thời gian*

Có thể thấy hiệu suất của RT-Tree không được tốt trong các truy vấn thời gian và lượng bộ nhớ yêu cầu bởi RT-Tree là quá lớn.

### 6.3. R-tree 3 chiều:

R-tree 3 chiều coi thời gian là một chiều bổ sung để biểu diễn các hình chữ nhật hai chiều với khoảng thời gian như các hộp ba chiều.

Phương pháp này giả định cả hai đầu mút của khoảng thời gian $$[t\_{start}, t\_{end})$$ cho mỗi hình chữ nhật đều biết và cố định. Nếu không biết thời gian kết thúc $$t\_{end}$$ thì hướng tiếp cận này sẽ không hiệu quả. Ví dụ: một đối tượng kéo dài từ một thời điểm cố định trong quá khứ cho tới nay. Một cách tiếp cận đó là biểu diễn thời điểm hiện tại bằng một thời điểm đủ xa trong thương lai. Điều này dẫn đến quá nhiều hộp và hiệu suất kém. Các phương pháp truy suất không gian tiêu chuẩn, chẳng hạn như R-tree không phù hợp để xử lý các đối tượng mở và mở rộng, trừ khi tất cả các chuyển động đều được biết trước.

Cây R-3D đã được so sánh với việc duy trì một chỉ mục không gian và thời gian riêng biệt, và kết quả cho thấy rằng hệ thống hợp nhất của một cây R-3D đơn là ưu việt cho các truy vấn không gian-thời gian, trong khi quyết định phụ thuộc vào độ tốt của các phép toán cho tải công việc kết hợp. Cây R-3D đã được đánh giá phân tích và thực nghiệm bằng cách sử dụng các tập dữ liệu tổng hợp.

### 6.4. R-tree 2 + 3:

Một giải pháp để giải quyết vấn đề hình học “mở” đó là duy trì một cặp hai cây R:

* Cây R 2D lưu trữ các mục hai chiều đại diện cho thông tin hiện tại (không gian) về dữ liệu
* Cây R 3D lưu trữ các mục ba chiều đại diện cho thông tin về quá khứ (không gian - thời gian)

Miễn là thời gian kết thúc $$t\_{end}$$ của đối tượng là không xác định, nó được lập chỉ mục bởi cây R 2D phía trước, giữ tọa độ bắt đầu của $$t\_{start}$$ cùng với ID của nó. Khi $$t\_{end}$$ đã xác định thì:

* Các mục liên quan được di chuyển từ cây R phía trước sang cây R ba chiều phía sau.
* Một mục mới lưu trữ vị trí hiện tại được cập nhật được chèn vào cây R phía trước.

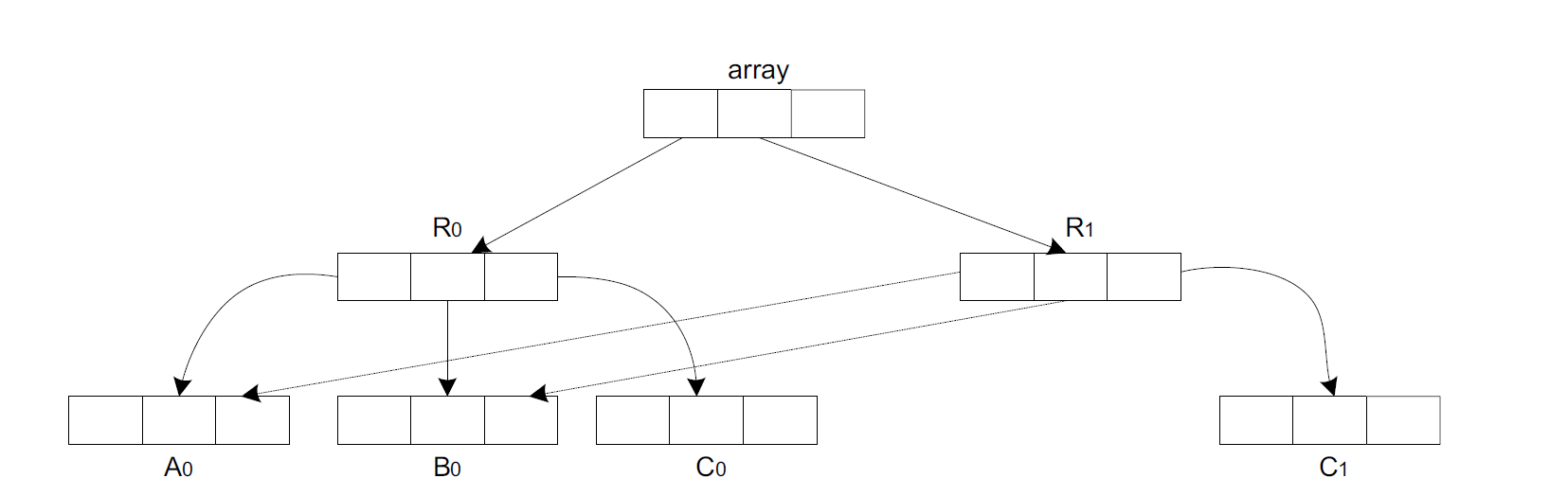
Nếu biết tất cả các chuyển động của đối tượng trước, cây R phía trước sẽ không được sử dụng, và cây R 2 + 3 sẽ giảm xuống thành cây R ba chiều được giới thiệu trước đó.

### 6.5. HR-tree:

Historical R-tree (Cây HR) là một phương pháp truy cập không gian thời gian dựa trên kỹ thuật chồng chéo. Khi có sự cập nhật, HR-tree tạo một cây R-tree mới, nhưng không thực tế để lưu trữ toàn bộ cây R-tree cho mỗi cập nhật. Do đó, HR-tree đại diện cho tất cả các cây R-tree chỉ trong logic dưới dạng một đồ thị không chu trình.

Với sự trợ giúp của một mảng trỏ đến gốc của các cây R-tree cơ bản, ta có thể dễ dàng truy cập cây R-tree mong muốn khi thực hiện truy vấn thời gian.

Khái niệm của cây chồng chéo đơn giản để hiểu và triển khai, và tối ưu không gian khi số lượng các đối tượng thay đổi vị trí trong không gian là tương đối nhỏ. Tuy nhiên, nếu số lượng đối tượng di chuyển từ một thời điểm đến thời điểm khác là lớn, phương pháp này sẽ trở thành các cây độc lập với nhau.



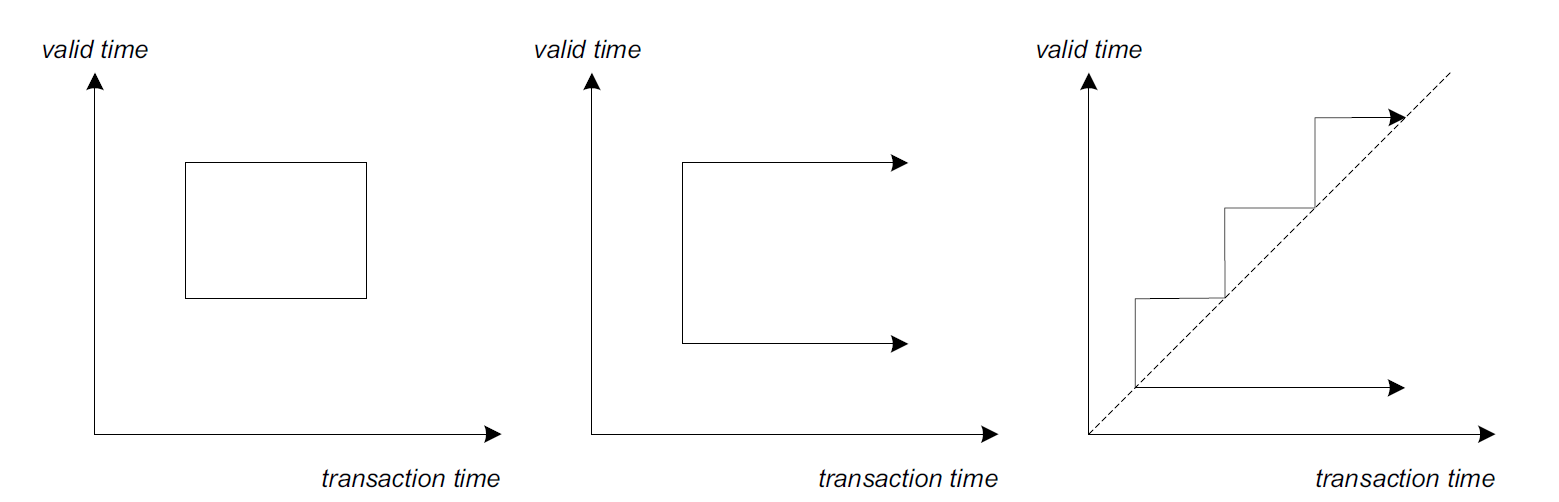
*Ví dụ về HR-tree*

Trong một so sánh hiệu suất đã được báo cáo giữa HR-tree, 2+3 R-tree và 3D R-tree, sử dụng các bộ dữ liệu tổng hợp do trình tạo dữ liệu GSTD tạo ra thì HR-tree được cho là hiệu quả hơn so với hai phương pháp còn lại đối với các truy vấn lát thời gian, trong khi điều ngược lại là đúng đối với các truy vấn khoảng thời gian. Ngoài ra, HR-tree thường dẫn đến một cấu trúc khá lớn do sao chép node, ngay cả đối với những thay đổi nhỏ trong dữ liệu.

### 6.6. Rst-tree:

Cây RST là một cấu trúc chỉ mục dữ liệu có thể xử lý dữ liệu không gian-lưỡng thời gian với hai chiều thời gian và hai chiều không gian. Dữ liệu có cả thời gian hợp lệ (quá khứ, hiện tại, tương lai khi dữ liệu đúng trong thực tế được mô hình hoá) và thời gian giao dịch được ghi lại gọi là lưỡng cực thời gian.

Đặc điểm độc đáo của dữ liệu thời gian là khoảng thời gian liên quan đến đối tượng liên quan đến thời điểm hiện tại, có nghĩa là các đầu mút của chúng liên quan chặt chẽ đến thời gian hiện tại. Ví dụ, khi có một nhân viên mới được tuyển dụng, dữ liệu cá nhân sẽ còn hợp lệ từ một thời điểm bắt đầu cụ thể cho đến thời điểm hiện tại, và điều tương tự cũng áp dụng cho thời gian giao dịch. Cây RST có thể xử lý các vùng khác nhau được tạo ra bởi sự kết hợp thuộc tính thời gian hợp lệ và thời gian giao dịch một cách hiệu quả hơn so với cây R-tree 3D.



*Các vùng ví dụ được tạo bởi thuộc tính thời gian hợp lệ và thời gian giao dịch*

### 6.7. PPR-tree:

Partially Persistent R-tree (PPR-tree), được thiết kế cho các ứng dụng về không gian và thời gian, trong đó các cập nhật được thực hiện theo thứ tự thời gian và chỉ có thể được thực hiện trên phiên bản cơ sở dữ liệu được ghi nhận cuối cùng.

PPR-tree là một đồ thị acyclic có hướng của các node với nhiều gốc, mỗi gốc đảm nhiệm việc ghi nhận một phần tiếp theo của sự tiến hóa R-tree tạm thời. Các bản ghi đối tượng được lưu trữ trong các node lá của PPR-tree và duy trì sự tiến hóa của các đối tượng dữ liệu R-tree tạm thời. Mỗi bản ghi dữ liệu bao gồm các trường thời gian chèn và xóa, và các node cây nội bộ duy trì sự tiến hóa của các mục chỉ mục tương ứng.

Một mục lá được coi là “còn sống” cho tất cả các thời điểm trong khoảng thời gian tương ứng. node lá hoặc node bên trong được coi là còn sống nếu không có sự phân chia nào được thực hiện trên node đó. PPR-tree được thiết kế để cải thiện hiệu quả nhập xuất bằng cách tập trung các mục sống vào một số node giới hạn.

Cấu trúc được cập nhật bằng cách thực hiện một chuỗi các hoạt động cập nhật trong các thời gian được chỉ định, trong đó chỉ có các mục sống tại thời điểm cập nhật được xem xét. Việc chèn và xóa có thể dẫn đến các thay đổi cấu trúc của PPR-tree. Cụ thể hơn trong quá trình chèn, node lá có thể bị đầy, do đó không thể thêm mục mới. Hiệu ứng này được gọi là tràn node. Sau khi xoá, số lượng mục còn sống trong một node có thể nhỏ hơn ngưỡng quy định. Cả hai trường hợp yêu cầu các điều chỉnh cấu trúc như thực hiện phân chia trên node lá tương ứng.

Với các thuật toán cho xử lý phạm vi và hàng xóm gần nhất, qua thực nghiệm ta thấy PPR-tree hiệu quả hơn so với các phương pháp trước đó.

### 6.8. MV3R-tree

MV3R-tree là một phương pháp truy cập được đề xuất để giải quyết các hạn chế của 3D R-tree và HR-tree. Nó bao gồm một cây R-tree đa phiên bản và một cây 3D R-tree được xây dựng trên các lá của cây R-tree đa phiên bản. Cây R-tree đa phiên bản phù hợp cho các truy vấn timestamp, trong khi cây 3D R-tree được thiết kế cho các truy vấn khoảng thời gian.

Quá trình xây dựng MV3R-tree sử dụng các heuristic khác nhau để xử lý việc chèn và xóa. Các mục nhập trong mỗi node chứa MBR, con trỏ đến node cấp cao hơn và các thời điểm chèn và xóa. MV3R-tree hiệu quả cho cả các truy vấn timestamp và khoảng thời gian và có yêu cầu không gian tương đối nhỏ.

MV3R-tree đã được thử nghiệm thông qua các thử nghiệm mở rộng và đã được chứng minh là hiệu quả hơn so với HR-tree và 3D R-tree cho độ linh hoạt của tập dữ liệu lên tới 20%.

### 6.9. TB-tree:

Các phương thức truy cập như 3D R-tree và MV3R-tree cố gắng duy trì sự gần nhau về mặt không gian của các đối tượng, đây là thuộc tính cơ bản để đảm bảo hiệu quả I/O trong quá trình xử lý truy vấn. Tuy nhiên, những cấu trúc này không thể đảm bảo tính hiệu quả khi chúng ta yêu cầu lịch sử của một tập đối tượng. Điều này là do các đoạn đường thuộc cùng một quỹ đạo có thể không gần nhau trong không gian dữ liệu và do đó chúng sẽ không được đặt trong cùng một trang lá trong quá trình xây dựng chỉ mục.

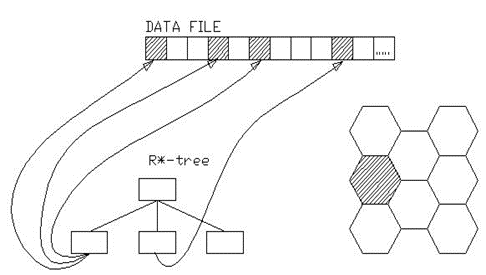
TB-tree (Trajectory Bundle) được thiết kế để vượt qua giới hạn này bằng cách nới lỏng tính chất cơ bản của phương pháp truy cập không gian và bảo tồn các quỹ đạo của các đối tượng. TB-tree lưu trữ các đoạn đường thuộc cùng một quỹ đạo trong cùng một node lá, bỏ qua sự gần nhau về mặt không gian. Các node lá thuộc cùng một quỹ đạo được liên kết thông qua một danh sách liên kết kép để truy xuất toàn bộ quỹ đạo của một đối tượng.

Nghiên cứu cho thấy rằng TB-tree vượt trội hơn so với các phương pháp truy cập khác cho các truy vấn liên quan đến lịch sử của các đối tượng, nhưng hiệu suất của nó thấp hơn đối với các truy vấn liên quan đến tọa độ không gian và thời gian của các đối tượng. Tuy nhiên, TB-tree vẫn là một phương pháp truy cập cạnh tranh.

### 6.10. Scalable and Efficient Trajectory Index (SETI)

SETI (Scalable and Efficient Trajectory Index) là một phương pháp để lập chỉ mục cho quỹ đạo và phân tách chiều không gian và thời gian. SETI chia nhỏ không gian thành các phân vùng cố định không trùng lặp và xây dựng một sparse index trên chiều thời gian cho mỗi phân vùng.

Cấu trúc của chỉ mục SETI được mô tả trong hình dưới đây. Mỗi phân vùng chứa các phân đoạn quỹ đạo rơi vào khu vực của nó.



*Cấu trúc SETI*

Các phân đoạn quỹ đạo được chèn vào các bộ dữ liệu, với mỗi trang dữ liệu chứa các phân đoạn quỹ đạo thuộc cùng một phân vùng. Vòng đời của một trang dữ liệu được xác định bằng khoảng thời gian tối thiểu bao phủ tất cả các phân đoạn quỹ đạo được lưu trữ trên trang đó. Tất cả các trang dữ liệu tương ứng với một phân vùng không gian được lập chỉ mục bằng R∗-tree.

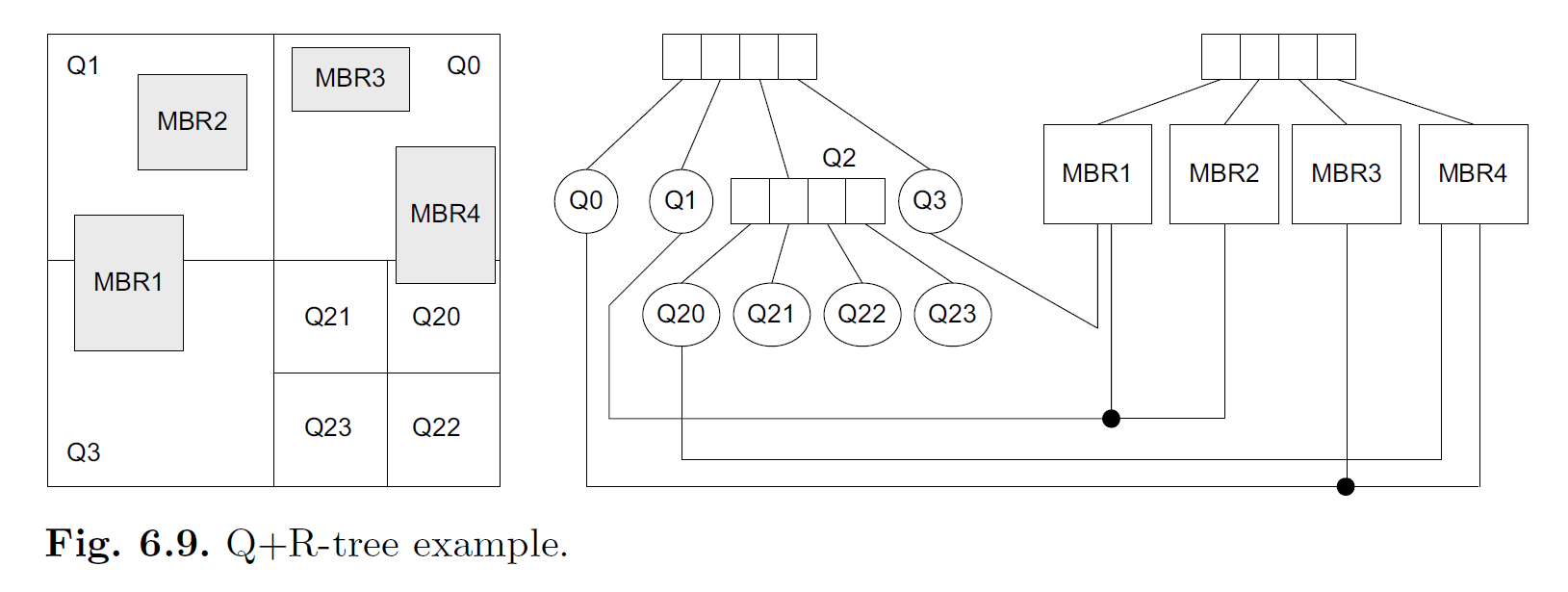
Với cơ chế này, SETI không bị ảnh hưởng bởi vấn đề chiều dài không gian. SETI cũng có lợi thế là một chỉ mục có thể được dễ dàng xây dựng trên một kỹ thuật lập chỉ mục hiện có, chẳng hạn như R-trees, và do đó được thực hiện dễ dàng hơn so với trường hợp thực hiện một kế hoạch lập chỉ mục vật lý.

### 6.11. Q+R-tree

Q+R-tree là một phương pháp chỉ mục hóa cho các đối tượng di chuyển giúp giảm thời gian cập nhật. Phương pháp này dựa trên quan sát rằng hầu hết các đối tượng di chuyển đều ở trạng thái bán tĩnh và được hướng dẫn bởi các ràng buộc địa hình.

Q+R-tree được xây dựng theo ba giai đoạn:

1. Một R∗-tree được xây dựng để xử lý các đối tượng bán tĩnh, được tổ chức dựa trên các đặc trưng địa hình của không gian dữ liệu.
2. Một Quadtree được sử dụng để tổ chức các đối tượng không ở trạng thái bán tĩnh.
3. Topography-based R∗-tree và Quadtree được kết hợp để tạo ra một phương pháp truy xuất duy nhất - Q+R-tree. Các MBR của R∗-tree có thể chồng chéo với các phân vùng trong Quadtree, và một danh sách liên kết có thể được sử dụng để chứa các con trỏ đến các MBR chồng chéo của R∗-tree.

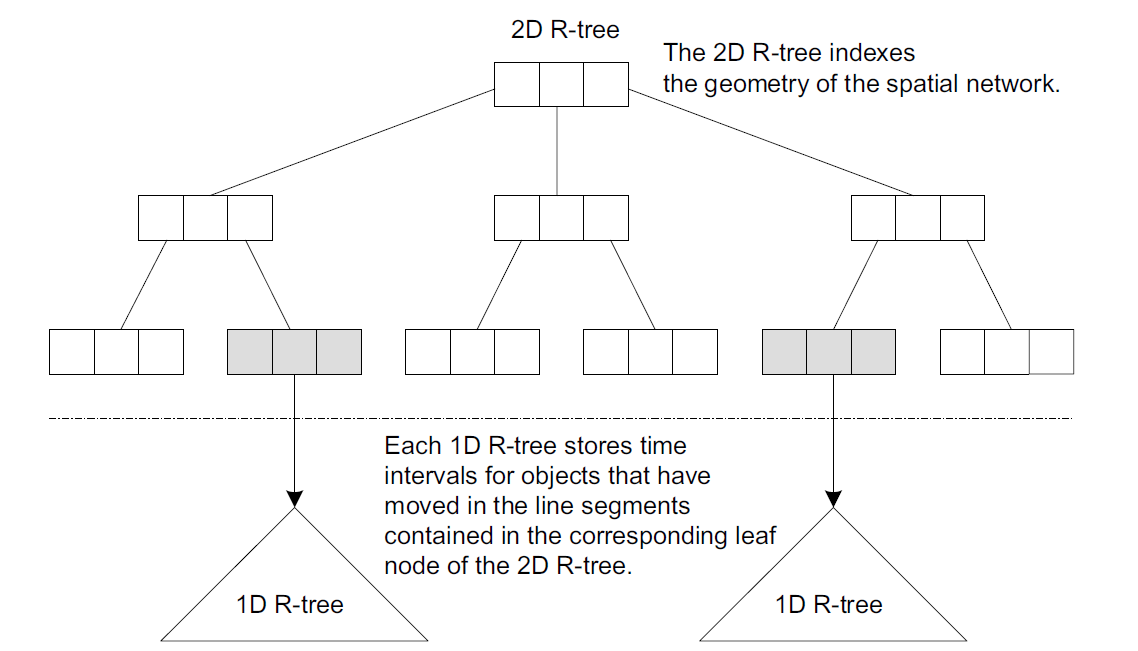


### 6.12. FRN-tree và MON-tree

Các phương pháp truy cập không gian thời gian đã được mô tả trước đó coi các đối tượng có thể di chuyển tự do mà không có ràng buộc, tuy nhiên, trong một số lĩnh vực ứng dụng, các đối tượng phải tuân thủ các ràng buộc không gian. Ví dụ, các đối tượng di chuyển trên mạng đường phải tuân thủ các quy tắc cụ thể được áp dụng bởi mạng không gian.

Một biến thể của cây R được gọi là cây R cố định mạng (FNR-tree) xem xét đến mạng không gian cơ sở và sử dụng một rừng các cây R để lập chỉ mục quỹ đạo của các đối tượng. FNR-tree là một phương pháp truy cập hai giai đoạn bao gồm một cây R 2D sắp xếp một tập hợp các cây R 1D.

Mỗi nút lá cây R 2D chứa các mục có dạng (lineID, MBR, orientation), trong đó lineID là ID của đoạn mạng không gian, MBR là MBR của đoạn mạng, và orientation là một cờ mô tả hình học chính xác của đoạn mạng trong MBR. Mỗi nút lá của cây R 1D lưu trữ các mục có dạng (oID, lineID, $$t\_{start}$$, $$t\_{end}$$, direction), trong đó oID là ID của đối tượng di chuyển, lineID là ID của đoạn mạng, $$t\_{start}$$ là thời điểm mà đối tượng di chuyển đã nhập vào đoạn mạng tương ứng, $$t\_{end}$$ là thời điểm mà đối tượng di chuyển đã rời khỏi đoạn mạng, và direction là một cờ chỉ ra hướng di chuyển.

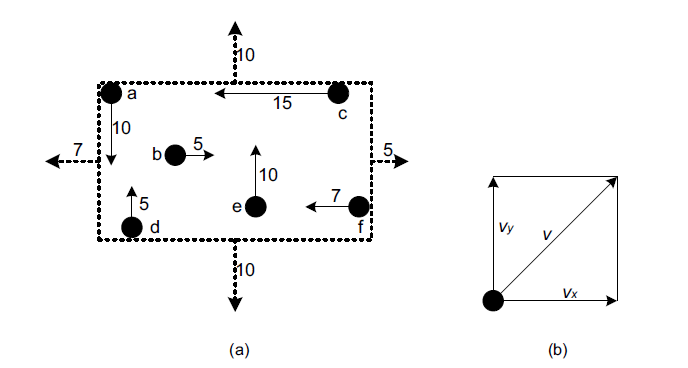


*Ví dụ về FNR-tree*

FNR-tree vượt trội hơn cây R 3D cho các truy vấn khoảng không gian thời gian và truy vấn lát thời gian trong hầu hết các trường hợp. Một biến thể của FNR-tree gọi là MON-tree sử dụng một cây R 2D cho mỗi đa giác của mạng không gian, nhưng không có kết quả đánh giá hiệu suất được đưa ra cho phương pháp này.

### 6.13. R-tree tham số hoá thời gian

Phương pháp truy cập TPR-tree (Time-Parameterized R-tree) cung cấp truy cập đến giá trị hiện tại và tương lai của các đối tượng dựa trên khái niệm MBR không tĩnh của các nút lá và nút bên trong. MBR của một đối tượng hoặc một nút cây là một hàm của thời gian và vector vận tốc của mỗi đối tượng được biết đến để xác định vị trí đối tượng hiện tại trong không gian và dự đoán vị trí của nó trong tương lai.



1. *Ví dụ về HCN giới hạn di chuyển, (b) Ánh xạ vector vận tốc tới các trục x và y*

Các MBR bao phủ của các nút nội bộ trong TPR-tree hiếm khi là tối thiểu, điều này có nghĩa là không gian trống lớn hơn so với cây R tĩnh lưu trữ đối tượng. Các thuật toán cho việc chèn, xóa và phân chia nút tương tự như các biến thể R-tree, ngoại trừ việc tích phân được sử dụng để xác định diện tích và mở rộng trùng lắp của MBR node. TPR-tree dựa trên khái niệm thời gian ngưỡng, chỉ ra khoảng thời gian mà cấu trúc vẫn giữ được hình dạng tốt mà không cần tái cấu trúc.

Hai biến thể khác của TPR-tree là STAR-tree và TPR ∗-tree, cho thấy hiệu suất tốt hơn trong xử lý truy vấn.

RPPF-tree (Past Present and Future R-tree) lập chỉ mục các vị trí quá khứ, hiện tại và tương lai của các đối tượng di chuyển bằng cách sử dụng hình chữ nhật giới hạn thời gian và khái niệm bền vững một phần.

### 6.14. VCI R-tree

Velocity Constrained Indexing R-tree (VCI R-tree) được điều chỉnh để thực hiện các thao tác hiệu quả trên vận tốc của đối tượng di chuyển. Sơ đồ quản lý này tránh được chi phí cập nhật của đối tượng di chuyển bằng cách sử dụng khái niệm tốc độ tối đa của mỗi đối tượng di chuyển.

VCI R-tree là một chỉ mục dựa trên cây R thông thường, được bổ sung bởi một trường bổ sung $$V\_{max}$$ trong mỗi nút để lưu trữ tốc độ tối đa được phép trên tất cả các đối tượng được phủ bởi nút này.

Khi một truy vấn khoảng cách được đưa ra cho một khoảng thời gian tương lai, MBR được mở rộng để đảm bảo bao phủ các đối tượng được phủ bởi node. Tuy nhiên, quá trình này có thể trả về một số cảnh báo sai. Để loại bỏ chúng, cần có giai đoạn xử lý hậu kỳ để phản ánh chính xác hơn các vị trí của các đối tượng di chuyển và giảm số lượng cảnh báo sai.

### 6.15. Tổng kết

Mục tiêu của hệ thống cơ sở dữ liệu không gian thời gian là quản lý các đặc tính không gian và thời gian của dữ liệu để đảm bảo hiệu quả trong việc quản lý và xử lý các truy vấn không gian thời gian. Tuy nhiên, việc sử dụng thông tin thời gian gặp phải nhiều thách thức, và các phương pháp ban đầu tập trung vào việc lập chỉ mục cho các vị trí quá khứ và hiện tại của các đối tượng.

Gần đây, các đề xuất đã giải quyết việc lập chỉ mục cho các vị trí tương lai của các đối tượng di chuyển, cho phép xử lý truy vấn dự đoán. Dự kiến sẽ có các kỹ thuật hiệu quả hơn được đề xuất trong tương lai do tính quan trọng của cơ sở dữ liệu không gian thời gian trong các dịch vụ dựa trên vị trí và các hệ thống giám sát.

## 7. R-tree trong đa phương tiện, lưu trữ và khai thác

Các ứng dụng hiện đại như đa phương tiện, lưu trữ và khai thác yêu cầu tổ chức và thao tác với các đối tượng phức tạp, đồng thời đặt ra yêu cầu đa dạng đối với việc xử lý truy vấn hiệu quả.

### 7.1. R-tree trong cơ sở dữ liệu đa phương tiện

Hệ thống quản lý cơ sở dữ liệu đa phương tiện nhằm biểu diễn và truy xuất hiệu quả đối tượng đa phương tiện như văn bản, hình ảnh, âm thanh và video. Đặc điểm cơ bản của đối tượng đa phương tiện khiến cho việc xử lý các truy vấn gặp nhiều khó khăn là:

* Nội dung phong phú, dẫn đến yêu cầu lưu trữ lớn và thuật toán quản lý phức tạp.
* Khó khăn trong việc miêu tả độ tương tự giữa hai đối tượng đa phương tiện yêu cầu áp dụng các thuật toán mới để tìm kiếm cơ sở dữ liệu đa phương tiện dựa trên nội dung.

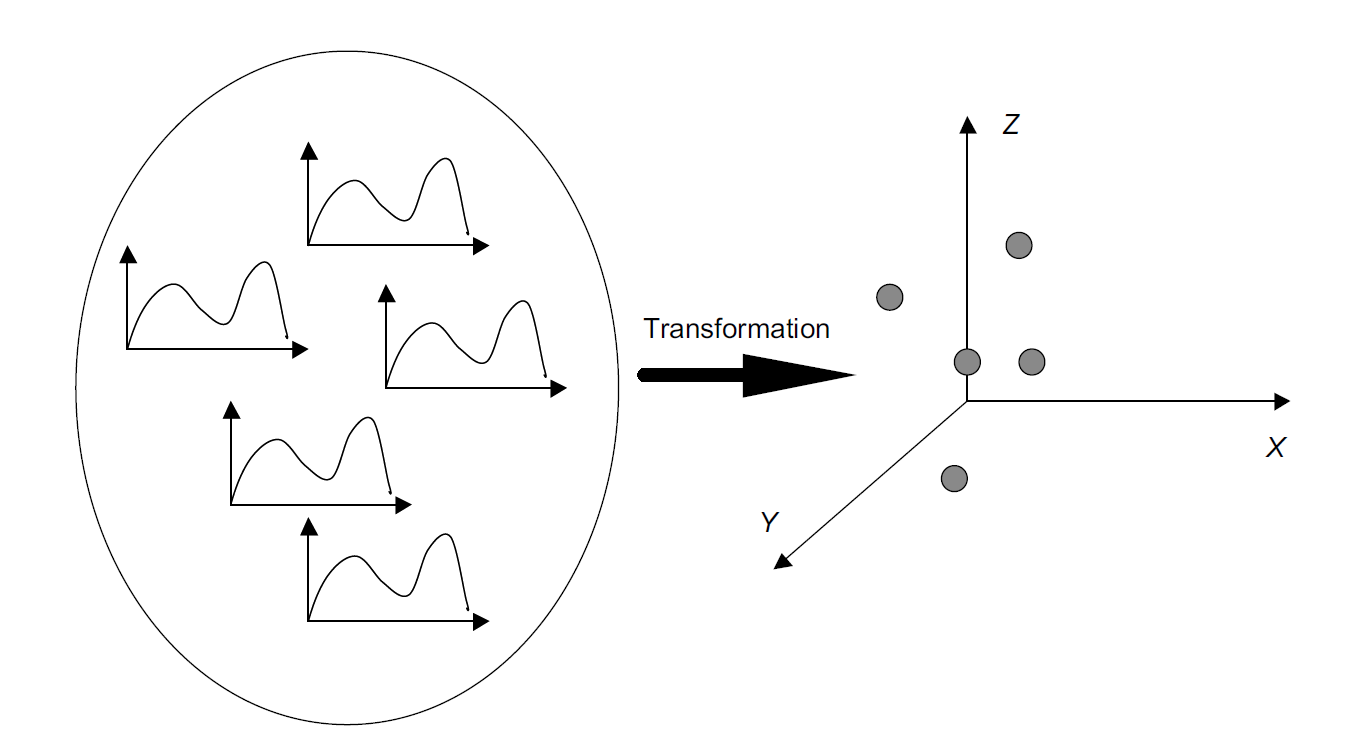
Trong bài viết, các kỹ thuật quan trọng đã được đề xuất để xử lý và điều khiển đối tượng đa phương tiện một cách hiệu quả. Đặc biệt, tập trung vào các phương pháp truy cập để xử lý truy vấn độ tương tự một cách hiệu quả. Ý tưởng cơ bản là chuyển đổi đối tượng đa phương tiện thành các vector đa chiều, cho phép sử dụng các phương pháp truy cập đa chiều để truy xuất hiệu quả.

#### 7.1.1. Lập chỉ mục đa phương tiện chung (GEMINI)

GEMINI là một phương pháp để lập chỉ mục các đối tượng đa phương tiện để tìm kiếm các loại đa phương tiện theo nội dung. Mục tiêu là tìm tất cả các đối tượng trong một khoảng cách nhất định từ một đối tượng truy vấn.

Phương pháp này sử dụng hai thành phần:

1. Một bước kiểm tra ban đầu "nhanh và đơn giản" để loại bỏ hầu hết các đối tượng không đáp ứng truy vấn,
2. Tìm kiếm thông qua một phương thức truy cập không gian (SAM) để tránh các hạn chế của tìm kiếm tuần tự.



*Ánh xạ chuỗi thời gian sang vector đa chiều*

GEMINI có thể được áp dụng để lập chỉ mục dữ liệu tín hiệu một chiều bằng cách trích xuất các đặc trưng từ mỗi chuỗi và ánh xạ chúng vào không gian đặc trưng. Việc lựa chọn các đặc trưng tốt là quan trọng để giảm thiểu sự sai sót và báo động sai.

Thành phần đầu tiên của GEMINI (kiểm tra nhanh và đơn giản) sử dụng các hệ số đầu tiên của biến đổi Fourier rời rạc (DFT) làm đặc trưng. Tính chất giới hạn dưới đảm bảo không có đối tượng nào bị bỏ sót trong câu trả lời cuối cùng.

Thành phần thứ hai của GEMINI sử dụng cấu trúc R-tree để lưu trữ các điểm ánh xạ trong không gian đặc trưng (gọi là F-index), giúp tìm kiếm nhanh hơn. Để thực hiện tìm kiếm phạm vi, hệ thống ánh xạ một truy vấn chuỗi thành một điểm trong không gian đặc trưng, sau đó sử dụng R-tree để lấy tất cả các điểm ứng cử viên trong không gian đặc trưng đã biến đổi mà nằm trong khoảng được chỉ định từ điểm truy vấn. Sau đó, hệ thống lấy các chuỗi tương ứng, tính toán khoảng cách thực tế từ điểm truy vấn và loại bỏ bất kỳ báo động sai nào.

GEMINI cũng có thể được sử dụng để trả lời các truy vấn hàng xóm gần nhất, nhưng điều này khó khăn hơn do khoảng cách giữa các đối tượng trong không gian đã biến đổi bị giảm. Một thuật toán được đề xuất để giải quyết vấn đề này, bao gồm thực hiện truy vấn k-NN trong không gian đã biến đổi, xác định khoảng cách an toàn, thực hiện truy vấn phạm vi hình tròn và loại bỏ các báo động sai để xác định k hàng xóm gần nhất trong không gian ban đầu.

Tuy nhiên, thuật toán này có thể tạo ra một số lượng lớn ứng viên, dẫn đến tăng chi phí CPU và I/O. Để giải quyết vấn đề này, một thuật toán tối ưu được đề xuất, thực hiện xử lý tăng dần các hàng xóm gần nhất và sử dụng tiêu chí kết thúc để tránh lấy các ứng cử viên không hứa hẹn. Bất kỳ biến thể R-tree nào cũng có thể được sử dụng với thuật toán này.

#### 7.1.2. Phương thức truy cập nhiều chiều

Phương pháp sử dụng các phương pháp truy cập không gian đã được áp dụng thành công vào các loại dữ liệu khác như âm thanh, hình ảnh màu sắc và video bằng cách sử dụng các phép biến đổi phù hợp.

Tuy nhiên, đối với các vector nhiều chiều đại diện cho các đối tượng đa phương tiện, hiệu suất của hầu hết các phương pháp truy cập không gian phân cấp giảm sút do sự tăng chiều không gian và sự trùng lặp quá mức của các node trung gian.

Để giải quyết vấn đề này, một số phương pháp lấy ý tưởng từ R-tree nhưng được điều chỉnh để xứ lý tốt hơn các vector nhiểu chiều.

**TV-tree** là một phương pháp được đề xuất bởi Lin, Jagadish và Faloutsos để xây dựng một cây sử dụng các chiều hứa hẹn để cắt tỉa và phân loại. Mỗi nút trong TV-tree đại diện cho vùng bao phủ tối thiểu (MBR) của tất cả các con cháu của nó, đó là một Lp-điểm cầu với trung tâm được đại diện bởi một vector telescopic và bán kính quyết định phạm vi của vùng.

TV-tree sử dụng một tập hợp các chiều hoạt động có thể phân biệt giữa các đối tượng và các chiều không hoạt động không thể phân biệt giữa các con cháu của nút. Thuật toán tìm kiếm của TV-tree tương tự như của phương pháp truy cập R-tree, và nó hỗ trợ các truy vấn liên kết không gian và lân cận nhất. Các phương pháp chèn, xóa và xử lý tràn của TV-tree đã được nghiên cứu kỹ lưỡng, và kết quả hiệu suất đã được báo cáo so sánh với phương pháp truy cập R∗-tree.

Cây tìm kiếm tương đồng (**SS-tree**) được đề xuất như một biến thể của phương pháp truy cập R-tree để xử lý dữ liệu có số chiều cao và cải thiện xử lý truy vấn tương đồng. Nó sử dụng khoảng cách Euclidean có trọng số làm độ đo tương đồng, có thể bắt được sự tương quan giữa các chiều và có thể được sử dụng ngay cả khi đã áp dụng một phép biến đổi tuyến tính cho các vector đặc trưng.

SS-tree tổ chức các vector đặc trưng theo cấu trúc phân cấp bằng cách xây dựng các vùng giới hạn, bao gồm tất cả các nút con. Hiệu suất của SS-tree được so sánh với R∗-tree bằng cách sử dụng các bộ dữ liệu phân bố đều và phân bố chuẩn, cũng như dữ liệu thực tế, và được cho là rất triển vọng với nhiều chỉ số hiệu suất khác nhau. Một biến thể cải tiến của SS-tree, gọi là SS+-tree, đã được đề xuất, sử dụng một vòng bao chặt hơn cho mỗi nút và thuật toán chia khác. Việc lựa chọn đúng các độ đo tương đồng và kiến thức chuyên môn có thể giúp điều chỉnh hiệu suất của SS-tree.

#### 7.1.3 R-tree và mô hình Markov ẩn trong truy xuất âm nhạc

Một ứng dụng cây R∗-tree là lập chỉ mục cho các mô hình Markov ẩn (HMMs) để truy xuất thông tin âm nhạc. HMMs được sử dụng để biểu diễn các bản nhạc, trong đó mỗi trạng thái là một chuyển tiếp âm nhạc được đại diện bằng một khoảng cách thời gian và một tỷ lệ IOI. Một tập hợp các bản nhạc được biểu diễn bằng các HMMs tương ứng, trong đó mỗi trạng thái được ánh xạ vào một hình chữ nhật 4 chiều. Để truy xuất thông tin từ một bản nhạc truy vấn, HMMs của nó được kiểm tra với tất cả các HMMs đã lưu bằng cách kiểm tra điểm 4 chiều của mỗi trạng thái của HMM truy vấn với hình chữ nhật 4 chiều của HMM đã lưu. Một cây R∗-tree được sử dụng để lập chỉ mục cho tất cả các HMM đã lưu để tăng tốc quá trình truy xuất thông tin. Kết quả cho thấy phương pháp này giảm thời gian thực thi truy vấn lên đến 7 lần mà không gây ra các kết quả sai.

#### 7.1.4. R-tree và bản đồ tự tổ chức

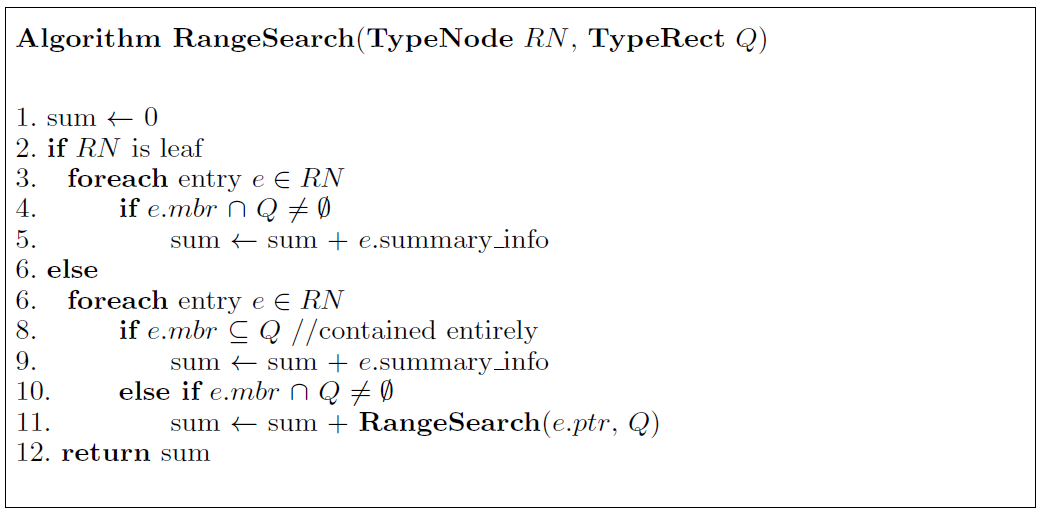
SOM-Based R∗-trees là một biến thể thú vị của R∗-trees kết hợp giữa Self-Organizing Maps (SOMs) và R∗-trees nhằm đạt được hiệu suất truy vấn tốt trong ứng dụng như tìm kiếm hình ảnh tương tự (hình ảnh được lập chỉ mục bằng đặc trưng). SOM là một dạng mạng thần kinh đưa dữ liệu từ không gian có nhiều chiều xuống không gian hai chiều, bằng cách duy trì đặc trưng của vectơ đặc trưng. Phép ánh xạ được gọi là bản đồ đặc trưng topo. Bản đồ đặc trưng topo được tạo ra bằng cách thực hiện quá trình đào tạo SOM. Kết quả là vector được tạo ra trên mỗi nút của SOM, được gọi là vector mã hóa. Sử dụng bản đồ đặc trưng topo, các hình ảnh tương tự được phân loại vào node gần nhất, phụ thuộc vào khoảng cách giữa vectơ đặc trưng của nó và bộ sưu tập các vectơ mã hóa. Đối với mỗi node, các hình ảnh tương tự đã được phân loại được tổ chức trong một danh sách hình ảnh phù hợp nhất (BMIL). Rõ ràng, tồn tại các nút trong SOM mà không có hình ảnh nào được phân loại, do đó chúng trống rỗng. Đối với xây dựng một R∗-tree, một điểm d chiều tương ứng với một vectơ mã hóa của bản đồ đặc trưng topo. Chúng ta kiểm tra tất cả các vectơ mã hóa. Các vectơ mã hóa tương ứng với các nút trống được bỏ qua. Đối với các vectơ mã hóa còn lại, các điểm tương ứng được chèn vào cây. Mỗi điểm được chèn vào nút lá mà trọng tâm của nó gần nhất với điểm cần chèn. Vì các nút trống được tránh, chỉ mục R∗-tree giúp định vị các nút không trống vào SOM. Do đó, tìm kiếm các hình ảnh tương tự theo cách này nhanh hơn so với việc phải kiểm tra toàn bộ SOM.

### 7.2. R-tree trong lưu trữ và khai thác dữ liệu

Kho dữ liệu là các cơ sở dữ liệu chuyên biệt đóng vai trò là nơi lưu trữ cho nhiều nguồn dữ liệu không đồng nhất, được tổ chức theo một lược đồ thống nhất để tạo thuận lợi cho việc đưa ra quyết định. Xử lý phân tích trực tuyến (OLAP), là một kỹ thuật được sử dụng trong kho dữ liệu để phân tích dựa trên khám phá. Ngược lại, khai thác dữ liệu là quá trình trích xuất thông tin hay mẫu từ cơ sở dữ liệu lớn một cách bán tự động. Cả lưu trữ và khai thác dữ liệu đều yêu cầu truy cập đến lượng dữ liệu lớn, điều này được tăng tốc bởi các chỉ mục.

Các biến thể R-tree được sử dụng cho OLAP và kho dữ liệu để lưu trữ thông tin tóm tắt trong các nút bên trong, giúp cho việc tìm kiếm nhanh hơn. Các biến thể R-tree này có thể được bổ sung thông tin tóm tắt, giúp tăng tốc độ tìm kiếm. Kết quả là cấu trúc được gọi là R∗a-tree. Các nút bên trong của R∗a-tree lưu trữ thông tin tóm tắt cho các chức năng tổng hợp phân tán như đếm và các hàm tổng hợp.

Thuật toán tìm kiếm theo phạm vi nhằm sử dụng thông tin tóm tắt để cải thiện hiệu suất. Nếu hình chữ nhật bao phủ tối thiểu (MBR) của một nút trong cây nội bộ nằm hoàn toàn trong vùng tìm kiếm, thì việc tìm kiếm đến cây con tương ứng sẽ được bỏ qua và kết quả sẽ được cập nhật thông qua thông tin tóm tắt được lưu trữ trong nút được phủ. Dưới đây là thuật toán tìm kiếm cho cây R∗a, giả định hàm tổng hợp tính toán được sử dụng.



Sự tiết kiệm từ việc sử dụng R∗a-tree cho các truy vấn OLAP tăng lên khi kích thước truy vấn lớn hơn vì các truy vấn lớn hơn có xu hướng hoàn toàn bao phủ nhiều hơn các MBR. Tuy nhiên, hiệu suất của R∗a-tree vẫn phụ thuộc vào kích thước truy vấn, và để giải quyết vấn đề này, Tao et al. đề xuất aP-tree để xử lý các truy vấn tổng hợp trên dữ liệu điểm phẳng. aP-tree ánh xạ các điểm từ không gian 2D sang các khoảng 1D và sử dụng một cấu trúc tương tự như MVB-tree để xử lý hiệu quả các truy vấn giao nhau cho các khoảng. Lợi thế của aP-tree so với R∗a-tree là chi phí truy vấn độc lập với kích thước cửa sổ, nhưng chi phí không gian cao hơn.

Các kỹ thuật khác, chẳng hạn như cây R-tree tổng hợp (aR-tree) và cây R-tree tổng hợp bộ nhớ ảo (VMAT), đã được đề xuất để lập chỉ mục cho các kho dữ liệu không gian. Ngoài ra, Roussopoulos et al. đã đề xuất lập chỉ mục các khối dữ liệu với một tập hợp các R-tree được đóng gói, gọi là cây cubtree, để xử lý các truy vấn hiệu quả.

Gần đây, R-tree cũng đã được sử dụng trong ngữ cảnh Khai phá dữ liệu, bao gồm các phương pháp cho phân cụm hiệu quả và tìm ra các mô hình không gian.

### 7.3. Tổng kết

Do đặc tính phức tạp của các đối tượng đa phương tiện, quản lý chúng là một công việc khó khăn. Hơn nữa, khái niệm về sự tương đồng giữa hai đối tượng đa phương tiện trong nhiều trường hợp có thể được coi là chủ quan. Những lý do trên yêu cầu các kỹ thuật tìm kiếm mới.

Nhu cầu truy cập vào các khối lượng dữ liệu lớn hiện diện trong cả ứng dụng lưu trữ dữ liệu và khai thác dữ liệu. Việc tăng tốc truy cập dữ liệu trong các môi trường này được đơn giản hóa bằng các chỉ mục chuyên dụng đã được đề xuất (ví dụ: chỉ mục bitmap). Tuy nhiên, các chỉ mục giống như R-tree vẫn tìm thấy các ứng dụng quan trọng.