ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC - KỸ THUẬT MÁY TÍNH

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT - MỞ RỘNG

Đề tài

KD - TREE và HASH

Giảng viên hướng dẫn: Nguyễn Đức Dũng
Sinh viên: Trần Tuấn Anh 2010878

TP. HỒ CHÍ MINH, THÁNG 1/2022

Mục lục

1 Đề tài

Mô tả: Trong chủ đề này, các bạn tìm hiểu về Kd-Tree, một cấu trúc dữ liệu nâng cao hỗ trợ việc tìm kiếm trên dữ liệu đa chiều. Các bạn hiện thực một Kd-Tree đơn giản với các giải thuật tìm hiểu được. Để đánh giá. Các bạn sử dụng cấu trúc dữ liệu Point3D float x, y, z; và sinh ngẫu nhiên các điểm trong không gian 3D, tầm giá trị trong khoảng [0...100] trên mỗi chiều, độ chính xác của các điểm là 0.001. Với việc tìm kiếm một điểm trong không gian này, các bạn có thể sử dụng giải thuật hash để băm một tọa độ vào các bin (kích thước có thể do mình tự chọn, ví dụ 0.1). Hãy so sánh hiệu năng của KdTree so với sử dụng kỹ thuật băm với số lượng điểm khác nhau. Khi số lượng điểm tăng lên, giải pháp nào tốt hơn, hãy phân tích và cho minh chứng.

Yêu cầu: các bạn làm một báo cáo đơn giản về các yêu cầu trong mô tả của tiểu luận. Nộp file .zip chứa báo cáo và mã hiện thực của mình.

2 Point3D và CutPlane

2.1 Point3D

- Biến thành viên: x, y, z.

- Phương thức:

+ Toán tử `[]` được tái định nghĩa để truy cập giá trị x, y, z bằng các số nguyên 0, 1, 2.

+ Phương thức tính khoảng cách Euclid giữa 2 điểm trong không gian Oxyz.

2.2 CutPlane

Cấu trúc dữ liệu CutPlane là một mặt phẳng trong không gian Oxyz có dạng :`ax + by + cz + d = 0`.

- Biến thành viên: gồm biến số nguyên lưu số chiều của không gian(mặc định là 3) và mảng số nguyên lưu hệ số của mặt phẳng.

- Phương thức:

+ Tính giá trị biểu thức `ax + by + cz + d` khi thay một điểm M(x0, y0, z0) vào biểu thức.

+ Tính khoảng cách Euclid giữa một điểm và mặt phẳng trong không gian Oxyz.

3 K-D Tree

-Trong khoa học máy tính, cây k-d là một cấu trúc dữ liệu phân vùng theo không gian để tổ chức các điểm trong không gian k chiều. cây k-d là một cấu trúc dữ liệu hữu ích cho một số ứng dụng. Trên cây, các node ở level khác nhau sẽ chia không gian thành 2 phần theo chiều khác nhau. Node bất kỳ sẽ có giá trị tại chiều không gian đang xét lớn hơn các node con bên trái và nhỏ hơn hoặc bằng các node con bên phải.

3.1 Biến thành viên

- Số chiều của không gian(mặc định là 3).
- Node gốc của cây.

3.2 Thêm một điểm vào cây

- Giải thuật: duyệt lần lượt từ gốc đến node lá thích hợp để thêm một điểm bất kỳ vào cây.
- Độ phức tạp thời gian:
+ Tốt nhất: O(logN), khi K-D Tree là cây cân bằng.
+ Tệ nhất: O(N), khi cây mất cân bằng.

3.3 Tìm điểm nhỏ nhất(tại chiều không gian xác định)

- Giải thuật:
+ Nếu chiều không gian đang xét tại Node giống với chiều không gian cần tìm, tìm điểm nhỏ nhất ở cây con bên trái.
+ Nếu chiều không gian đang xét tại Node khác với chiều không gian cần tìm, tiến hành tìm kiếm trên cả 2 cây con và so sánh với giá trị ở Node gốc.
- Độ phức tạp thời gian:
+ Xấp xỉ O(logN) khi cây cân bằng.
+ O(N) khi cây mất cân bằng.

3.4 Xóa điểm khỏi cây

- Giải thuật: + Duyệt từ gốc đến khi gặp được Node cần xóa
+ Nếu cây con bên phải tồn tại, tìm Node nhỏ nhất tại cây con bên phải với chiều không gian đang xét tại Node cần xóa. Thay giá trị của Node cần xóa bằng Node nhỏ nhất tìm được. Xóa node nhỏ nhất tìm được.
+ Nếu cây con bên trái tồn tại và cây con bên phải không tồn tại, tìm Node nhỏ nhất tại cây con bên trái với chiều không gian đang xét tại Node cần xóa. Thay giá trị của Node cần xóa bằng Node nhỏ nhất tìm được. Xóa node nhỏ nhất tìm được và thay cây con bên trái thành bên phải.
+ Nếu cả 2 cây con không tồn tại. Xóa Node cần xóa hiện tại.
- Độ phức tạp thời gian:
+ Xấp xỉ O(logN) khi cây cân bằng.
+ O(N) khi cây mất cân bằng.

3.5 Tìm điểm gần nhất với điểm cho trước

- Giải thuật:
+ Tìm điểm gần nhất ở cây con bên trái hoặc bên phải tùy theo giá trị tại chiều đang xét của điểm cho trước.
+ So sánh khoảng cách tìm được với giá trị tại Node đang xét.

+ Nếu khoảng cách từ điểm cho trước đến mặt phân cách giữa 2 cây con nhỏ hơn khoảng cách nhỏ nhất đã tìm được, cần phải duyệt cả bên cây con còn lại.

- Độ phức tạp:

+ Tốt nhất: O(logN).

+ Tệ nhất: O(N).

+ Trung bình: O(logN).

3.6 Tìm điểm lân cận điểm cho trước

- Giải thuật

+ Tìm các điểm nằm trong khoảng lân cận ở cây con bên trái hoặc bên phải tùy theo giá trị tại chiều đang xét của điểm cho trước.

+ Nếu khoảng cách từ điểm cho trước đến mặt phân cách giữa 2 cây con nhỏ hơn khoảng lân cận đã cho, cần phải duyệt cả bên cây con còn lại.

- Độ phức tạp:

+ Tốt nhất: O(logN).

+ Tệ nhất: O(N).

+ Trung bình: O(logN).

4 Locality Sensitive Hashing

Trong khoa học máy tính, băm nhạy cảm với địa phương là một kỹ thuật thuật toán băm các giá trị đầu vào tương tự vào cùng một "nhóm" với xác suất cao. Vì các điểm tương tự kết thúc trong cùng một nhóm, nên kỹ thuật này có thể được sử dụng để phân nhóm dữ liệu và tìm kiếm điểm gần nhất.

4.1 Biến thành viên

- Biến số nguyên là số lượng mặt phân cách.
- Tổng số khe có được với k mặt phân cách là 2^k.
- Số lượng điểm có trong mảng.
- Khoảng giá trị của các điểm(cận trên và cận dưới).
- Số chiều không gian sử dụng.
- Các mặt phân cách(được tạo ngẫu nhiên).
- Bảng dữ liệu.
- Có thể tăng hiệu quả trong việc tìm kiếm trong không gian(tránh việc 2 điểm gần nhau nằm ở 2 block khác nhau) bằng cách tạo nhiều bảng con và tạo ngẫu nhiên thêm các mặt phân cách tương ứng. Do đó, cần có biến số lưu số lượng bảng con đã tạo.

4.2 Hàm băm

- Giải thuật:

+ Mỗi mặt phân cách sẽ tạo ra 2 không gian con lần lượt là 0 và 1. Một điểm bất kỳ chỉ có thể nằm ở 1 trong 2 không gian con đó.

+ So sánh vị trí của điểm đang xét đối với từng mặt phân cách để tìm ra index của không gian con chứa điểm đó.

- Độ phức tạp: O(DK), với D là số chiều không gian, K là số lượng mặt phân cách trên 1 bảng.

4.3 Thêm 1 điểm vào bảng

- Giải thuật:

+ Khi thêm 1 điểm vào bảng, ta phải thêm điểm đó vào tất cả các bảng con.

+ Sử dụng hàm băm để tìm ra index của không gian con chứa điểm đó và thêm điểm vào không gian đó. Giải quyết đụng độ bằng kĩ thuật chainning. Hàm băm với bảng con khác nhau sẽ cho index khác nhau.

- Độ phức tạp: O(LDK), trong đó L là số lượng bảng con(là một hằng số), O(DK) là độ phức tap của hàm băm.

4.4 Xóa 1 điểm khỏi bảng

- Giải thuật:

+ Khi xóa 1 điểm khỏi bảng, ta phải xóa điểm đó trên tất cả các bảng con.

+ Sử dụng hàm băm để tìm ra index của không gian con chứa điểm đó và xóa điểm trên không gian đó. Hàm băm với bảng con khác nhau sẽ cho index khác nhau.

- Độ phức tạp: O(LDK), trong đó L là số lượng bảng con(là một hằng số), O(DK) là độ phức tạp của hàm băm.

4.5 Tìm điểm gần nhất

4.5.1 Giải thuật

- Băm điểm để tìm ra index của không gian chứa điểm. Duyệt qua tất cả các điểm nằm trong không gian đó để tìm điểm nhỏ nhất.

- Lặp lại với tất cả các bảng con.

4.5.2 Vấn đề về độ chính xác

- Với việc số lượng bảng con là 1 số hữu hạn, vẫn luôn có trường hợp giải thuật phía trên chưa thật sự chính xác.

- Giải thuật hỗ trợ:

+ Những mặt phân cách có khoảng cách từ mặt tới điểm đang tìm kiếm xa hơn khoảng cách nhỏ nhất đã tìm được sẽ là những bit không thể thay thế. Ngược lại những mặt có khoảng cách gần hơn sẽ là những bit có thể thay thế và ta phải xét mọi trường hợp thay thế được.(ví dụ: có k mặt phân cách có khoảng cách gần sẽ có 2^k không gian cần xét, bao gồm cả không gian ban đầu của điểm đang xét.

+ Duyệt qua tất cả bảng con để tìm bảng có số mặt phân cách nằm gần nhỏ nhất. Thực hiện kiểm tra lại với bảng con đó.

4.5.3 Độ phức tạp

$$O(LDK + LD\frac{N}{2^K} + 2^k Dk)$$

- Trong đó: L là số lượng bảng con, D là số chiều không gian, K là số mặt phân cách có trong 1 bảng và k là số mặt phân cách cần kiểm tra lại.
- Tệ nhất: O(NlogN), khi k = K = logN.
- Tốt nhất: O(logN), khi k = 0, K = logN.
- Trung bình: O(logN).

4.6 Tìm các điểm lân cận

4.6.1 Giải thuật

- Băm điểm để tìm ra index của không gian chứa điểm. Duyệt qua tất cả các điểm nằm trong không gian đó để tìm các điểm lân cận.
- Lặp lại với tất cả các bảng con.

4.6.2 Vấn đề về độ chính xác

- Với việc số lượng bảng con là 1 số hữu hạn, vẫn luôn có trường hợp giải thuật phía trên chưa thật sự chính xác.
- Giải thuật hỗ trợ:

+ Những mặt phân cách có khoảng cách từ mặt tới điểm đang tìm kiếm xa hơn khoảng lân cận sẽ là những bit không thể thay thế. Ngược lại những mặt có khoảng cách gần hơn sẽ là những bit có thể thay thế và ta phải xét mọi trường hợp thay thế được.(ví dụ: có k mặt phân cách có khoảng cách gần sẽ có 2^k không gian cần xét, bao gồm cả không gian ban đầu của điểm đang xét.

+ Duyệt qua tất cả bảng con để tìm bảng có số mặt phân cách nằm gần nhỏ nhất. Thực hiện kiểm tra lại với bảng con đó.

4.6.3 Độ phức tạp

$$O(LDK + LD\frac{N}{2^K} + 2^k Dk)$$

- Trong đó: L là số lượng bảng con, D là số chiều không gian, K là số mặt phân cách có trong 1 bảng và k là số mặt phân cách nhỏ nhất cần kiểm tra lại.
- Tệ nhất: O(NlogN), khi k = K = logN.
- Tốt nhất: O(logN), khi k = 0, K = logN.
- Trung bình: O(logN).

5 So sánh hiệu năng

5.1 Hiệu năng về không gian

- K-D Tree: O(1).
- LSH: O(L), với L là số bảng con. Có thể giải quyết vấn đề về không gian bằng cách sử dụng con trỏ để trỏ tới 1 dữ liệu duy nhất => Độ phức tạp không gian: O(1).

5.2 Hiệu năng về thời gian

	K-D Tree	LS Hash
INSERT	O(logN)->O(N)	O(logN)
REMOVE	O(logN)->O(N)	O(logN)
NN SEARCH	O(logN)->O(N)	O(logN)->O(NlogN)

Bảng 1: Độ phức tạp về thời gian

- Khi số điểm tăng lên, KD Tree sẽ dễ bị mất cân bằng dẫn đến độ phức tạp xấp xỉ O(n), trong khi cấu trúc Locality Sensitive Hash sẽ càng chính xác dẫn đến độ phức tạp trung bình thấp.

6 Kết luận

- Cả 2 cấu trúc dữ liệu là K-D Tree và Locality Sensitive Hash đều có những ưu và nhược điểm của riêng nó. Độ hiệu quả của một cấu trúc dữ liệu cũng như giải thuật còn tùy thuộc vào từng trường hợp riêng biệt và những yêu cầu mà thực tiễn đặt ra. Đòi hỏi người sử dụng phải nắm vững kiến thức cũng như linh hoạt trong việc ứng dụng kiến thức để giải quyết vấn đề.