ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

ĐỒ ÁN CUỐI KÌ
CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

CHỦ ĐỀ:
CẤU TRÚC DỮ LIỆU K-D TREE

Giảng viên lý thuyết: Nguyễn Thanh Sơn
Giảng viên hướng dẫn thực hành: Nguyễn Đức Vũ
Người thực hiện: Nguyễn Hoàng Hải - 21522034

--TP.Hồ Chí Minh, tháng 5 năm 2022--

I. Tổng quan đề tài

1. Lý do chọn đề tài

Bài toán phân loại có lẽ không còn xa lạ bởi sự phổ biến của nó trong lĩnh vực học máy cũng như ứng dụng của nó trong nhiều tác vụ thực tế. Một trong những thuật toán supervised-learning đơn giản thường được sử dụng để giải quyết bài toán này chính là thuật toán K-Nearest Neighbors (KNN), trong đó, nhãn của một điểm dữ liệu mới được suy ra từ nhãn của K điểm dữ liệu gần nhất trong tập dữ liệu huấn luyện. Thao tác quan trọng nhất và cũng tốn nhiều thời gian nhất trong thuật toán KNN chính là tìm K điểm từ tập điểm hữu hạn sao cho chúng gần nhất với một điểm cho trước trong một không gian. Với những bộ dữ liệu rất lớn, việc tìm ra một thuật toán tối ưu cho thao tác này sẽ làm tăng tính hiệu quả của thuật toán KNN lên nhiều lần.

Spatial Decomposition (hay chia nhỏ không gian) là một hướng tiếp cận tốt để giải quyết thao tác trên. Ở phương pháp này, không gian Euclidean được chia nhỏ thành nhiều phần, các phần của không gian được lưu trữ trừu tượng bằng một cấu trúc dữ liệu phù hợp. Từ đó, việc tìm kiếm tập hợp K điểm gần nhất không cần phải duyệt hết toàn bộ không gian mà chỉ cần xem xét những vùng không gian tiềm năng.

Theo hướng tiếp cận trên, K-d Tree là một cấu trúc dữ liệu vô cùng hiệu quả trong nhiều trường hợp của bộ dữ liệu. Cấu trúc dữ liệu K-d Tree cho phép lưu trữ không gian Euclideanan bằng cây nhị phân tìm kiếm mà mỗi nút (khác nút lá) sẽ đại diện cho một siêu phẳng chia không gian Euclidean thành 2 phần. Để có thể khai thác sâu hơn vào bài toán tìm tập điểm gần nhất cũng như trang bị một công cụ hiệu quả trong lập trình, em đã quyết định chọn đề tài “Cấu trúc dữ liệu K-d Tree”.

2. Mục tiêu nghiên cứu

- Mục tiêu chung: Mô tả cấu trúc dữ liệu K-d Tree và ứng dụng K-d Tree vào bài toán tìm điểm gần nhất

- Mục tiêu cụ thể:

  - Mô tả cấu trúc dữ liệu K-d Tree..
  - Chỉ rõ ưu điểm và hạn chế của cấu trúc dữ liệu K-d Tree.
  - Chỉ ra tiềm năng của cấu trúc dữ liệu K-d Tree.

3. Đối tượng nghiên cứu

Cấu trúc dữ liệu K-d Tree.

4. Phạm vi nghiên cứu

Cấu trúc dữ liệu K-d Tree trong bài toán tìm điểm gần nhất.

Cấu trúc dữ liệu K-d Tree trong khoa học máy tính.

II. Nội dung nghiên cứu

1. Cơ sở lý thuyết

a) Cấu trúc dữ liệu

Trong khoa học máy tính, cấu trúc dữ liệu là một cách lưu trữ dữ liệu trong máy tính sao cho nó có thể được sử dụng một cách hiệu quả. Mỗi loại cấu trúc dữ liệu sẽ cần một lượng bộ nhớ phù hợp cho việc lưu trữ cũng như thực hiện các thao tác trên bộ dữ liệu. Cấu trúc dữ liệu luôn gắn liền với các thao tác trên dữ liệu, mỗi cấu trúc dữ liệu sẽ cho phép một số thao tác dữ liệu được thực hiện với độ phức tạp tương ứng.

b) Cây là gì? Các khái niệm liên quan đến cây

Cây là một cấu trúc dữ liệu gồm một tập hữu hạn các nút, giữa các nút có quan hệ phân cấp gọi là quan hệ "cha-con", có một nút đặt biệt gọi là gốc. Trước hết, có một số khái niệm xoay quanh cây như sau:

- Cấp của nút là số con của một nút.
- Nút lá là nút có cấp bằng 0.
- Nút nhánh là nút không phải là nút lá.
- Cấp của cây là cấp cao nhất của một nút trên cây.
- Mỗi nút chỉ có một nút cha.
- Mức của một nút là I khi nút cha của nó có mức là i-1. Gốc của cây luôn có mức là 1.
- Chiều cao (hay chiều sâu) của một cây là số mức lớn nhất của nút có trên cây đó.

Xét theo lý thuyết đồ thị, cấu trúc dữ liệu cây có thể biểu diễn dưới dạng đồ thị một cách trực quan. Đồ thị G = (V, E) là một cây có các tính chất sau:

- G liên thông và không có chu trình.
- |V| = n-1.
- Nếu bỏ đi một cạnh bất kỳ thì G mất tính liên thông.
- Nếu bổ sung một cạnh nối hai đỉnh không kề nhau thì xuất hiện một chu trình duy nhất.
- Chỉ có duy nhất một đường đi đơn giữa hai đỉnh bất kỳ trong V.

Mọi đồ thị con liên thông của một cây cũng là một cây và được gọi là cây con. Nếu cây có gốc, một đỉnh u và tất cả các hậu duệ của nó được gọi là cây con gốc u.

c) Cây nhị phân, cây nhị phân cân bằng

Cây nhị phân là một dạng cấu trúc dữ liệu cây mà cấp của mỗi nút tối đa bằng hai. Với một nút trên cây, người ta cũng phân biệt cây con trái và cây con phải của nút đó. Cây con có gốc là đỉnh trái của một nút gọi là cây con trái của nút đó. Cây con phải của một nút là cây có gốc là nút con phải của nút đó. Mỗi cây con của cây nhị phân cũng là một cây nhị phân.

Cây nhị phân cân bằng là một cây nhị phân. Tại đó, với một nút bất kỳ không phải là lá trên cây, chiều cao của cây con trái và cây con phải của nút đó chênh lệch không quá một đơn vị. Mỗi cây con trong cây nhị phân cân bằng cũng là một cây nhị phân cân bằng.

d) Siêu phẳng

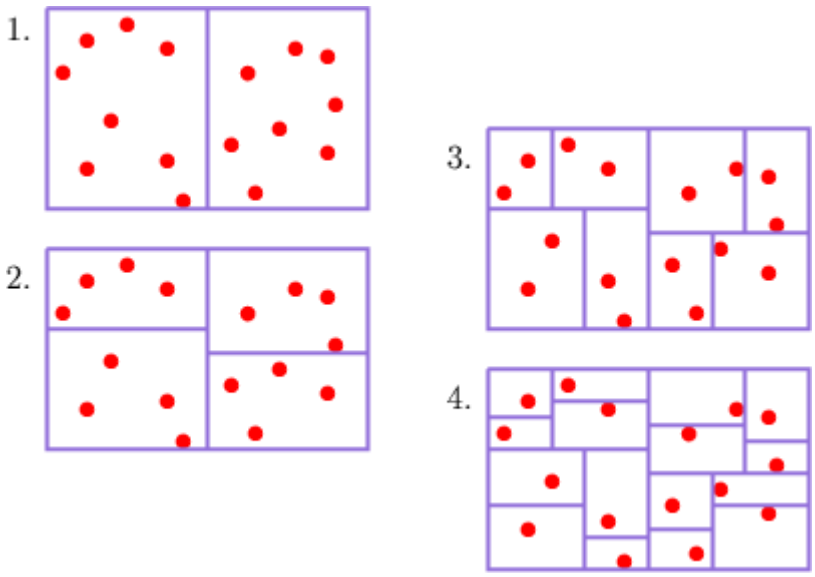
Xét một không gian Euclidean có *K* chiều, siêu phẳng là một không gian con mà số chiều của nó bằng *K-1*. Siêu phẳng chia không gian làm hai nửa không gian. Ví dụ trong không gian 3 chiều, mỗi siêu phẳng sẽ là một mặt phẳng. Cụ thể hơn, Xét không gian 3 chiều với hệ trục tọa độ Oxyz, các siêu phẳng sẽ là các mặt phẳng $x = a, y = b, z = c$ (Với *a, b, c* là các tham số).

e) Cấu trúc dữ liệu K-d Tree

Cấu trúc dữ liệu K-d Tree là một cây nhị phân mà dữ liệu của mỗi nút trên cây là một điểm trong không gian Euclidean K chiều. Cây K-d Tree được xây dựng như sau:

- Với một tập các điểm trong không gian Euclidean đa chiều, chọn ra một chiều không gian *x* rồi tạo một tập *A* chứa các giá trị tọa độ có được bằng cách chiếu lần lượt các điểm này lên không gian *x*. Sau đó, sắp xếp tập A theo thứ tự tăng dần, chọn điểm ứng với giá trị là trung vị của tập A làm nút gốc, các điểm ứng với giá trị nằm bên trái trung vị sẽ thuộc cây con trái, các điểm ứng với giá trị nằm bên phải trung vị sẽ thuộc cây con phải.
- Các cây con trái và cây con phải của nút gốc tiếp tục được xây dựng theo cách trên.

Như vậy cây K-d Tree cũng là một cây nhị phân cân bằng. Mỗi nút khác lá trên cây đóng vai trò phân chia tập điểm thành hai nửa bằng cách chọn ra một siêu phẳng mà từ đó cây con trái của nút này chứa các điểm nằm trong cùng một phía so với siêu phẳng đó, các điểm thuộc phía còn lại sẽ thuộc cây con phải.



Không gian 2D bị phân cắt bởi các siêu phẳng

2. Các thao tác dữ liệu xoay quanh cây K-d Tree

a) Dựng cây K-d Tree

Có rất nhiều cách lựa chọn siêu phẳng cho việc chia nhỏ không gian, do đó, có rất nhiều phương pháp để xây dựng cây K-d Tree. Trong số đó, có một phương pháp phổ biến chính là chọn tuần hoàn lần lượt các chiều không gian theo một thứ tự nhất định. Từ không gian *x* được chọn, thiết lập một siêu phẳng *x* = *a* sao cho siêu phẳng này chia tập điểm trong dữ liệu thành hai nửa nằm về hai phía của nó. Ví dụ trong không gian 2 chiều với hệ trục tọa độ Oxy, với 4 điểm trong tập dữ liệu là {(1; 2), (2; 3), (2; 4), (3; 1)}:

- Xét cây con có nút gốc ở mức 1, chiều không gian đầu tiên được xét đến sẽ là “x”, tiến hành sắp xếp các điểm dữ liệu trên theo hoành độ thì ta có được tập {(1; 2), (2; 3), (2; 4), (3; 1)}.

- Siêu phẳng sẽ là x = 2 (giá trị trung vị), điểm được chọn làm nút gốc là (2; 3), tập điểm thuộc cây con trái là {(1; 2)}, tập điểm thuộc cây con phải là {(2; 4), (3; 1)}.
- Xét cây con có nút gốc ở mức 2, chiều không gian được chọn sẽ là “y”, do cây con trái chỉ có một nút nên chỉ cần tiến hành sắp xếp các điểm dữ liệu ở cây con phải theo tung độ, ta có được tập {(3;1), (2; 4)}.
- Siêu phẳng tiếp theo sẽ là y = 1 (chọn điểm gần giữa), chọn (3; 1) làm nút gốc, cây con trái sẽ rỗng, tập điểm của cây con phải là {(2; 4)}.

Như vậy, để biểu diễn *N* điểm trong không gian, cây K-d Tree cần bộ nhớ lưu trữ là O(*N*) và độ phức tạp khi dựng cây là O(*N* $\log^2 N$).

(Vẽ hình)

Sau đây là phần code minh họa việc xây dựng cây K-d Tree theo cách trên với *points* là tập điểm, *dim* là số chiều.

```
def make_kd_tree(points, dim, i=0):
    if len(points) > 1:
        points.sort(key=lambda x: x[i])
        i = (i + 1) % dim
        half = len(points) >> 1
        return [
            make_kd_tree(points[: half], dim, i),
            make_kd_tree(points[half + 1:], dim, i),
            points[half]
        ]
    elif len(points) == 1:
        return [None, None, points[0]]
```

b) Tìm một điểm gần nhất

Việc tìm ra điểm gần nhất là thao tác dữ liệu rất quan trọng sau khi xây dựng cây K-d Tree từ tập điểm ban đầu vì đây chính là

thao tác cốt lõi thể hiện tính hiệu quả của cấu trúc dữ liệu này. Với một đỉnh *A* nằm trong cùng không gian Euclidean với các điểm dữ liệu trong K-d Tree, thao tác tìm điểm gần *A* nhất được thực hiện như sau:

- Bắt đầu bằng việc chọn nút gốc của cây K-d Tree.
- (1) Tính khoảng cách giữa điểm *A* và điểm được đại diện bởi nút đang chọn để cập nhật giá trị của nút gần *A* nhất tính đến hiện tại.
- Với chiều không gian *x* được chọn để thiết lập siêu phẳng tại nút đang chọn, gọi *a, b* lần lượt là giá trị tọa độ của điểm *A* và điểm mà nút đang chọn chứa khi chiếu chúng lên *x*.
- Nếu $a > b$ thì tiến hành chọn nút gốc của cây con phải rồi quay về bước (1), sau đó kiểm tra xem khoảng cách từ *A* siêu phẳng mà nút đang chọn đại diện có nhỏ hơn khoảng cách giữa *A* và điểm gần nhất được tìm thấy tính đến hiện tại hay không, nếu có thì tiếp tục chọn nút gốc của cây con trái rồi quay về bước (1).
- Nếu $a < b$ thì tiến hành chọn nút gốc của cây con trái rồi quay về bước (1), sau đó kiểm tra xem khoảng cách từ *A* siêu phẳng mà nút đang chọn đại diện có nhỏ hơn khoảng cách giữa *A* và điểm gần nhất được tìm thấy tính đến hiện tại hay không, nếu có thì tiếp tục chọn nút gốc của cây con phải rồi quay về bước (1).

Như vậy, trong quá trình tìm điểm gần *A* nhất, ta có thể bỏ qua rất nhiều điểm không cần xét đến, điều này làm tăng tính hiệu quả của thuật toán này. Bên cạnh đó, tính chất cân bằng của

cây K-d Tree cũng giúp cho việc thực hiện thao tác này hiệu quả hơn. Thao tác này có độ phức tạp trung bình là O(log N).

```
def get_nearest(kd_node, point, dim, dist_func, i=0, best=None):
    if kd_node is not None:
        dist = dist_func(point, kd_node[2], dim)
        if not best:
            best = [dist, kd_node[2]]
        elif dist < best[0]:
            best[0], best[1] = dist, kd_node[2]

        dx = kd_node[2][i] - point[i]
        #dx < 0 nghĩa là điểm P nằm phía bên phải của siêu phẳng x
= kd_node[2][i] (nằm về phía x > kd_node[2][i])
        order = [] #thứ tự duyệt
        if dx < 0:
            order = [1, 0] #duyệt nhánh phải trước, nhánh trái sau
        else:
            order = [0, 1] #duyệt nhánh trái trước, nhánh phải sau

        #Duyệt nhánh cây ưu tiên trước
        get_nearest(kd_node[order[0]], point, dim, dist_func, (i +
1) % dim, best)
        #Nếu cần thì duyệt nhánh cây còn lại
        '''
        Giải thích:
            Nếu khoảng cách nhỏ nhất đã tìm ra (tính đến thời điểm
hiện tại) bé hơn khoảng cách xét trên siêu phẳng i giữa điểm P và
điểm kd_node[2]
            thì ta bỏ qua việc xét điểm này (nghĩa là bỏ qua việc
xét nhánh cây có gốc là điểm đó)
        '''
        dx = abs(dx)
        if dx < best[0]:
            get_nearest(kd_node[order[1]], point, dim, dist_func,
(i + 1) % dim, best)
    return best
```

c) Tìm *m* điểm gần nhất

Việc truy vấn tìm *m* điểm gần nhất với một điểm *A* cũng được thực hiện tương tự như thao tác tìm một điểm gần nhất, tuy

nhiên, thao tác này cần một cấu trúc dữ liệu heap phục vụ cho việc lưu trữ *m* điểm gần nhất cũng như chọn ra điểm xa nhất trong *m* điểm đã tìm được để đối sánh với khoảng cách giữa *A* và siêu phẳng nhằm kiểm tra các tập điểm tiềm năng cho việc duyệt cây K-d Tree.

d) Thêm một nút vào cây

Thao tác thêm một đỉnh vào cây K-d Tree có độ phức tạp O(log). Với *A* là điểm cần thêm vào, thao tác này được tiến hành như sau:

- Bắt đầu chọn từ nút gốc của K-d Tree.
- (1) Với chiều không gian x mà siêu phẳng chia hai nửa không gian được thiết lập từ nút đang chọn là $x = a$ (*a* là giá trị có được khi chiếu điểm được chọn lên không gian một chiều *x*), chiếu điểm của nút đang chọn và điểm *A* lên chiều không gian *x* ta lần lượt nhận được hai giá trị tọa độ là *a* và *b*.
- Nếu $a > b$ thì *A* thuộc về cây con trái của nút đang chọn, nếu cây con trái có từ một nút trở lên, tiếp tục chọn nút gốc của cây con trái và thực hiện lại từ bước (1).
- Nếu $a <= b$ thì *A* thuộc về cây con phải của nút đang chọn, nếu cây con phải có từ một nút trở lên, tiếp tục chọn nút gốc của cây con phải và thực hiện lại từ bước (1).
- Nếu cả hai cây con đều rỗng, có nghĩa là nút được chọn là nút lá:
  + Nếu $a > b$, gán *A* là nút con bên trái của nút đang chọn.
  + Nếu $a <= b$, gán *A* là nút con bên phải của nút đang chọn.

Tuy nhiên, việc thực hiện thao tác thêm nút vào cây K-d Tree có thể khiến cây mất tính cân bằng, làm giảm hiệu quả khi thực hiện các thao tác khác trên K-d Tree.

```
def add_node(kdtree, point, dim, i=0):
    if (kdtree == None):
        kdtree = [None, None, point]
        return
    if (kdtree[2][i] > point[i]):
        i = (i + 1) % dim
        add_node(kdtree[0], point, dim, i)
    else:
        i = (i + 1) % dim
        add_node(kdtree[1], point, dim, i)
```

e) Xóa một nút khỏi cây

Thao tác xóa một nút được thực hiện lần lượt qua 2 bước: tìm nút trên cây, tìm nút thay thế.

Với điểm *A* là điểm cần xóa, thao tác tìm một nút trên cây được thực hiện như sau:

- Bắt đầu chọn từ nút gốc của K-d Tree.
- (1) Với chiều không gian x mà siêu phẳng chia hai nửa không gian được thiết lập từ nút đang chọn là $x = a$ (*a* là giá trị có được khi chiếu điểm được chọn lên không gian một chiều *x*), chiếu điểm của nút đang chọn và điểm *A* lên chiều không gian *x* ta lần lượt nhận được hai giá trị tọa độ là *a* và *b*.
- Nếu $a > b$ thì *A* thuộc về cây con trái của nút đang chọn, nếu cây con trái có từ một nút trở lên, tiếp tục chọn nút gốc của cây con trái và thực hiện lại từ bước (1), nếu không thì dừng việc tìm kiếm ở nhánh cây con này.
- Nếu $a < b$ thì *A* thuộc về cây con phải của nút đang chọn, nếu cây con phải có từ một nút trở lên, tiếp tục chọn nút gốc

của cây con phải và thực hiện lại từ bước (1), nếu không thì dừng việc tìm kiếm ở nhánh cây con này.

- Nếu *a* = *b* thì kiểm tra xem *A* có phải là nút đang chọn không, nếu phải thì tiến hành bước sau, nếu không thì *A* có thể thuộc về cây con trái hoặc cây con phải nên phải tiến hành chọn nút gốc của cây con trái rồi thực hiện lại từ bước (1) nếu cây con trái có từ một nút trở lên, sau đó tiến hành chọn nút gốc của cây con phải rồi thực hiện lại từ bước (1) nếu cây con phải có từ một nút trở lên.
- Nếu nút cần xóa là nút lá thì xóa ngay nút đó, nếu không thì thực hiện việc tìm kiếm nút thay thế.

Thao tác tìm nút thay thế:

- Với cây con có gốc là nút chứa dữ liệu của điểm cần thay th và *x* là chiều không gian được chọn tại nút này để thiết lập siêu phẳng cho việc chia nhỏ không gian, tìm điểm *B* thuộc cây con phải của *A* sao cho nó có giá trị tọa độ khi chiếu lên *x* là nhỏ nhất, thay *A* bằng *B*, sau đó tiến hành thao tác xóa điểm *B*.

f) Ưu điểm và hạn chế của cây K-d Tree

K-d Tree là một cấu trúc dữ liệu vô cùng vượt trội:

- Dễ cài đặt
- Thời gian thực hiện thao tác tìm kiếm nhanh hơn rất nhiều so với thuật toán ngây thơ

Bên cạnh đó, K-d Tree cũng có những hạn chế:

- Thao tác thêm hoặc xóa nút khỏi cây khiến cây mất tính cân bằng, ảnh hưởng hiệu suất của các thao tác khác.
- Hiệu suất của các thao tác trên cây K-d Tree giảm rất nhiều đối với những bộ dữ liệu điểm trong không gian nhiều chiều.

- Hiệu suất của thao tác tìm kiếm điểm gần nhất giảm rõ rệt khi điểm được truy vấn cách quá xa so với những điểm trong K-d Tree.

III. Kết luận và kiến nghị

Bên cạnh những hạn chế, các thao tác trên K-d Tree được thực hiện rất nhanh chóng so với thuật toán ngây thơ và trả về kết quả chính xác. Do đó, K-d Tree là một cấu trúc dữ liệu vô cùng hiệu quả trong bài toán tìm điểm gần nhất. Ngoài ra, K-d Tree cũng là tiền đề cho nhiều cấu trúc dữ liệu như BKD Tree ứng dụng cho bài toán tìm kiếm ảnh, CKD Tree ứng dụng trong bài toán phân lớp hình ảnh và rất nhiều cấu trúc dữ liệu hiệu quả khác.

IV. Tài liệu tham khảo

- https://github.com/jyotipmahes/Implementation-of-ML-algos-in-Python/blob/master/Nearest%20Neighbors%20Search.ipynb
- Lê, H. M. (1999). Cây. In *Cấu trúc dữ liệu và giải thuật* (pp. 70–77). book, Đại học Sư phạm Hà Nội.
- https://en.wikipedia.org/wiki/K-d_tree
- https://github.com/Vectorized/Python-KD-Tree/blob/master/kd_tree.py
- https://www.programiz.com/dsa/balanced-binary-tree#:~:text=A%20balanced%20binary%20tree%2C%20also,node%2C%20visit%20Tree%20Data%20Structure.
- https://en.wikipedia.org/wiki/Hyperplane#:~:text=In%20geometry%2C%20a%20hyperplane%20is,are%20the%201%2Ddimensional%20lines.
- https://en.wikipedia.org/wiki/Euclideanan_space
- http://www.cs.utah.edu/~lifeifei/cis5930/kdtree.pdf

- https://kevinmacdonald.me/post/kdtree_comparison/