

## Nhóm 5 - Chủ đề 4

# FINDING SIMILAR ITEMS

Tìm kiếm tương tự

Thành viên nhóm:

|                 |          |
|-----------------|----------|
| Hoàng Hữu Đức   | 23020046 |
| Lê Minh Tuấn    | 23020149 |
| Phạm Minh Thông | 23020164 |
| Bùi An Huy      | 23020079 |
| Lê Minh Đạt     | 23020037 |

# Mục lục

1. Giới thiệu
2. Ứng dụng thực tế
3. Biểu diễn dữ liệu
4. Các phép đo độ tương đồng
5. Các kỹ thuật tìm cặp tương đồng

# Mục lục

1. Giới thiệu

2. Ứng dụng thực tế

3. Biểu diễn dữ liệu

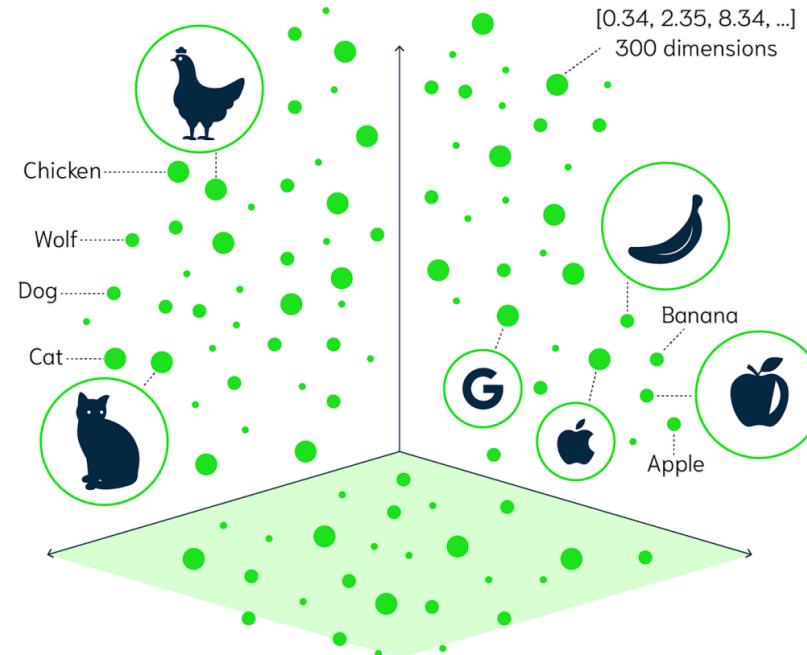
4. Các phép đo độ tương đồng

5. Các kỹ thuật tìm cặp tương đồng

# Giới thiệu



Hình 1: For?



Hình 2: Present



Hình 3: Find

# Mục lục

1. Giới thiệu

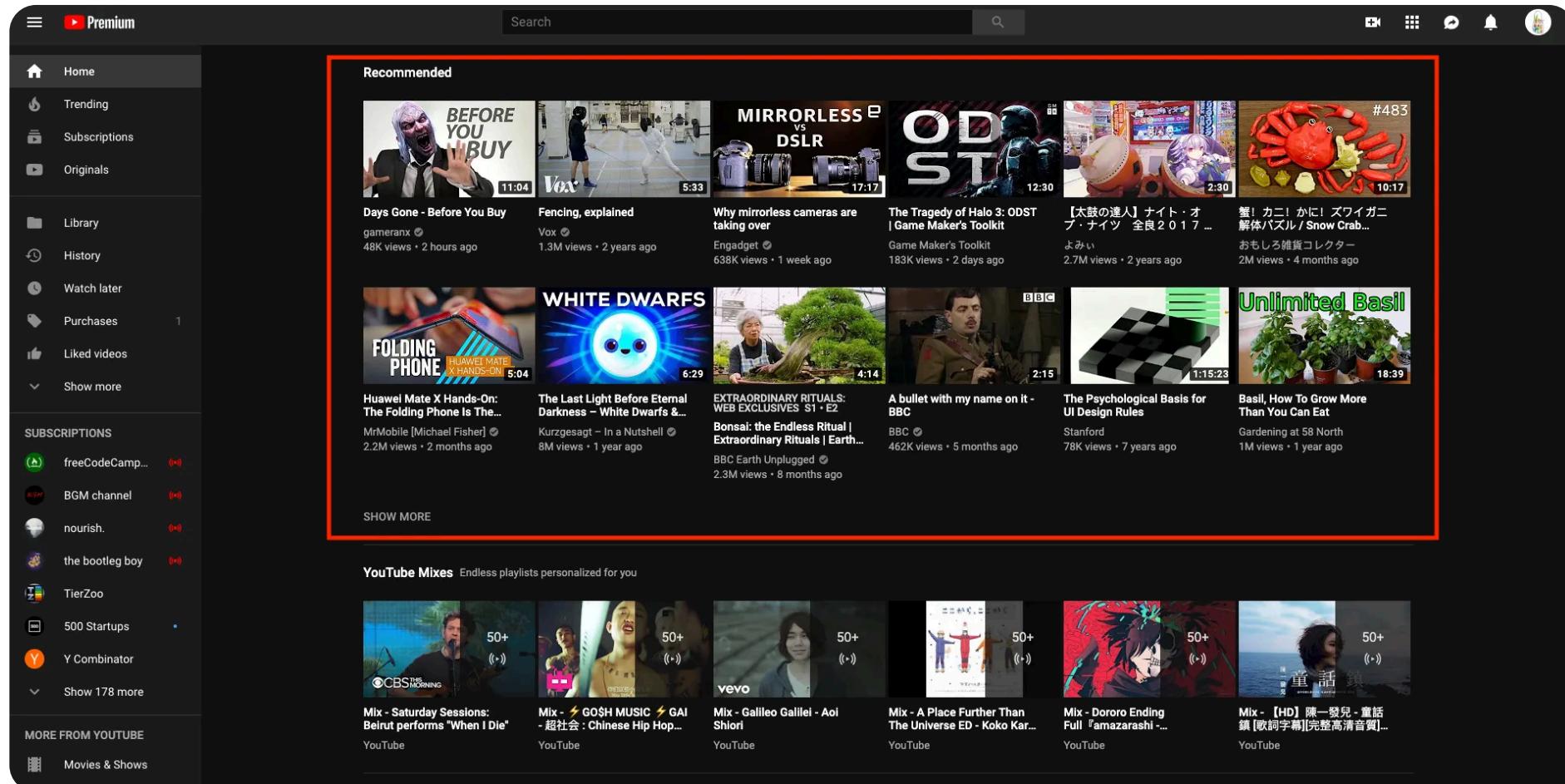
2. Ứng dụng thực tế

3. Biểu diễn dữ liệu

4. Các phép đo độ tương đồng

5. Các kỹ thuật tìm cặp tương đồng

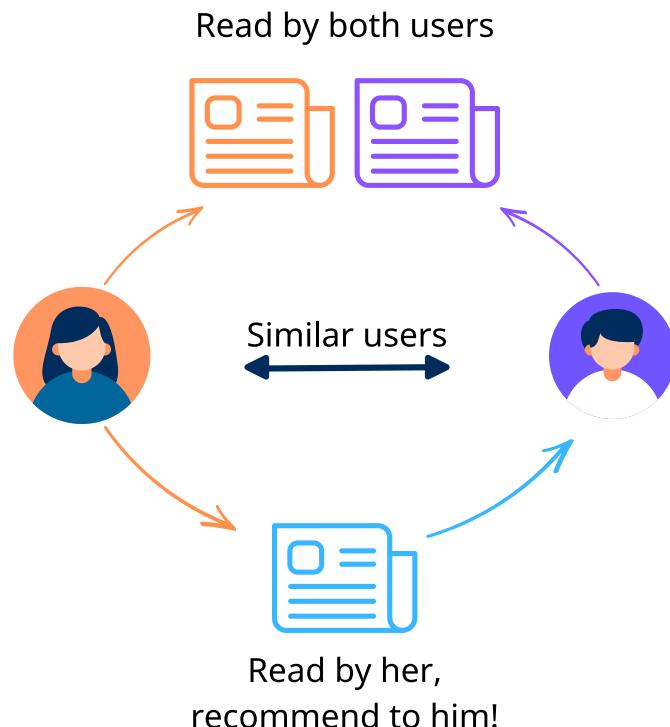
## 2.1. Hệ thống đề xuất



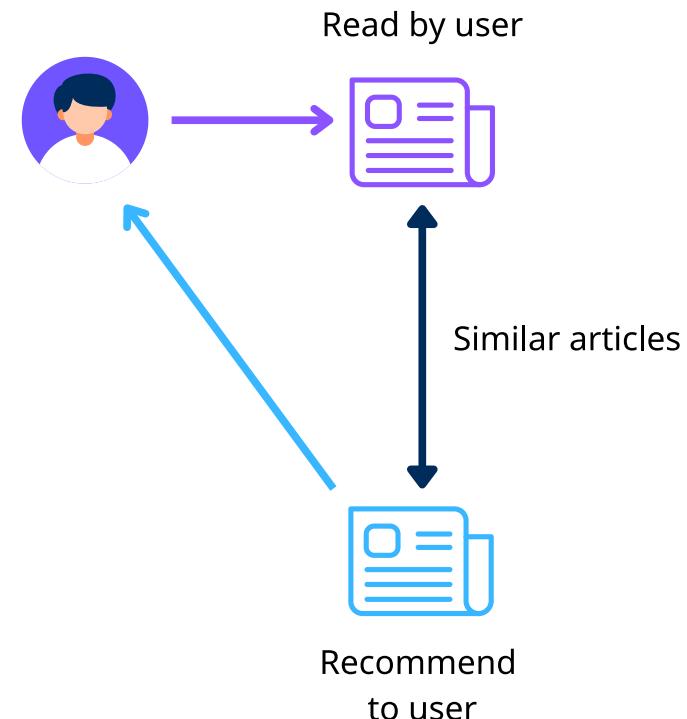
Hình 4: Hệ thống đề xuất trên YouTube

## 2.1. Hệ thống đề xuất

### COLLABORATIVE FILTERING



### CONTENT-BASED FILTERING



Hình 5: Các loại hệ thống đề xuất

## 2.2. Nhận diện trùng lặp văn bản

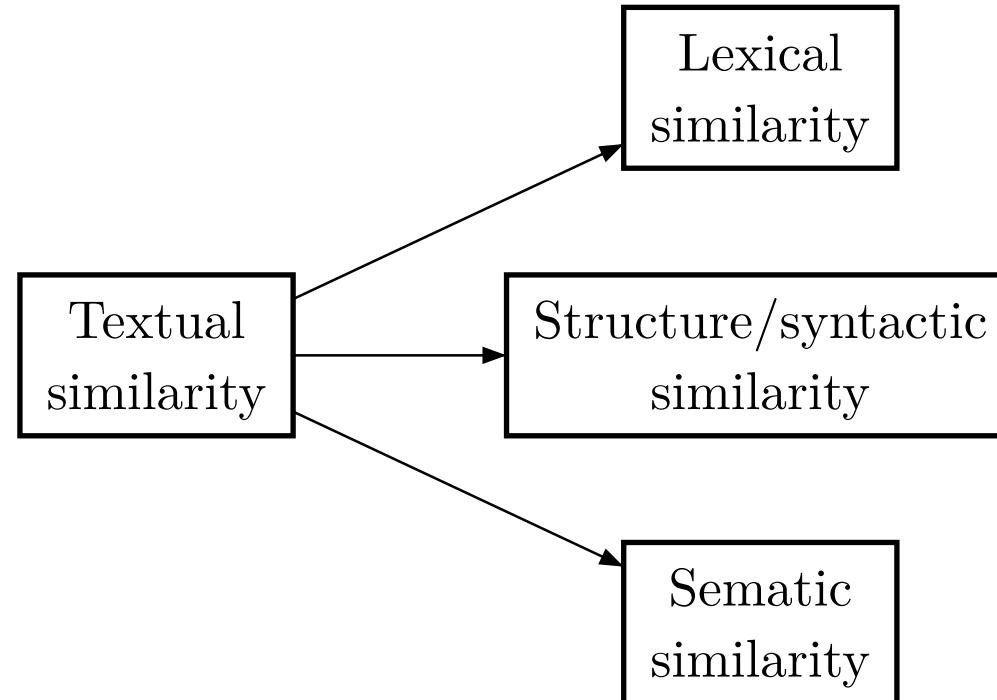
The screenshot shows the Turnitin software interface. On the left, a document titled "Exegesis paper.docx" is displayed. The first section is "INTRODUCTION", which discusses Romans 8:12-17 as one of the most remarkable paragraphs in the New Testament, quoting Paul's teaching on the Spirit's work in believers. The second section is "TEXT AND STRUCTURE", with a "Translation" section containing Greek text from Romans 8:12-17 and its English interpretation. At the bottom left are "Share" and "Search" buttons. The main area shows a similarity report with a "Top sources" tab selected. The overall similarity is 40%. The report lists eight sources with their respective percentages:

| Rank | Source  | Percentage |
|------|---|------------|
| 1    | dokumen.pub<br>INTERNET                             | 2%         |
| 2    | ebin.pub<br>INTERNET                                | 2%         |
| 3    | sbts-wordpress-uploads.s3.amazonaws.com<br>INTERNET | 1%         |
| 4    | bibliotekanauki.pl<br>INTERNET                      | 1%         |
| 5    | blogs.corban.edu<br>INTERNET                        | <1%        |
| 6    | ourarchive.otago.ac.nz<br>INTERNET                  | <1%        |
| 7    | repository.sbts.edu<br>INTERNET                     | <1%        |
| 8    | dartmouthbible.org<br>INTERNET                      | <1%        |

At the bottom right, it says "Page 1 of 26".

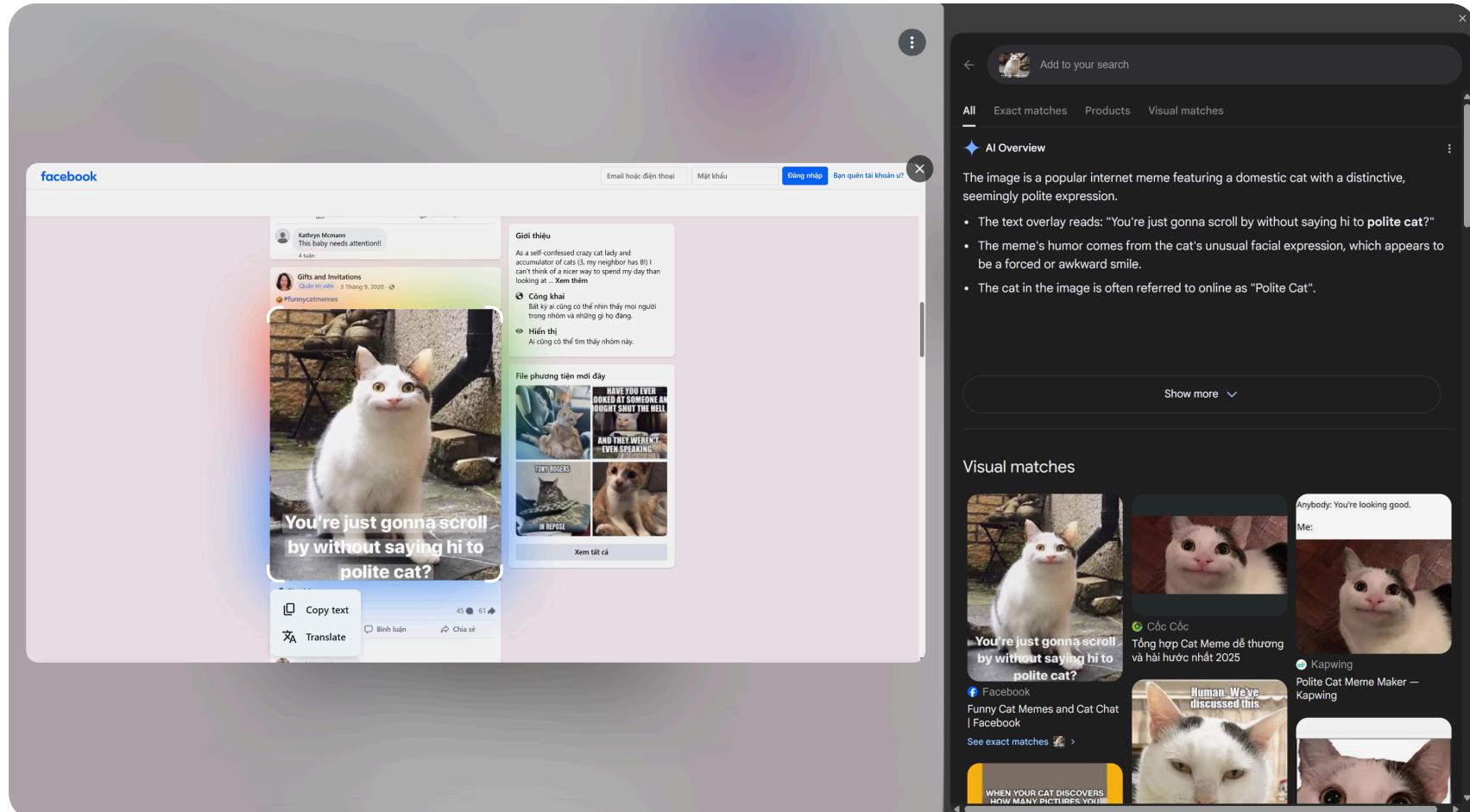
Hình 6: Giao diện phần mềm Turnitin

## 2.2. Nhận diện trùng lặp văn bản



Hình 7: Các loại tương đồng văn bản

## 2.3. Tìm hình ảnh giống nhau

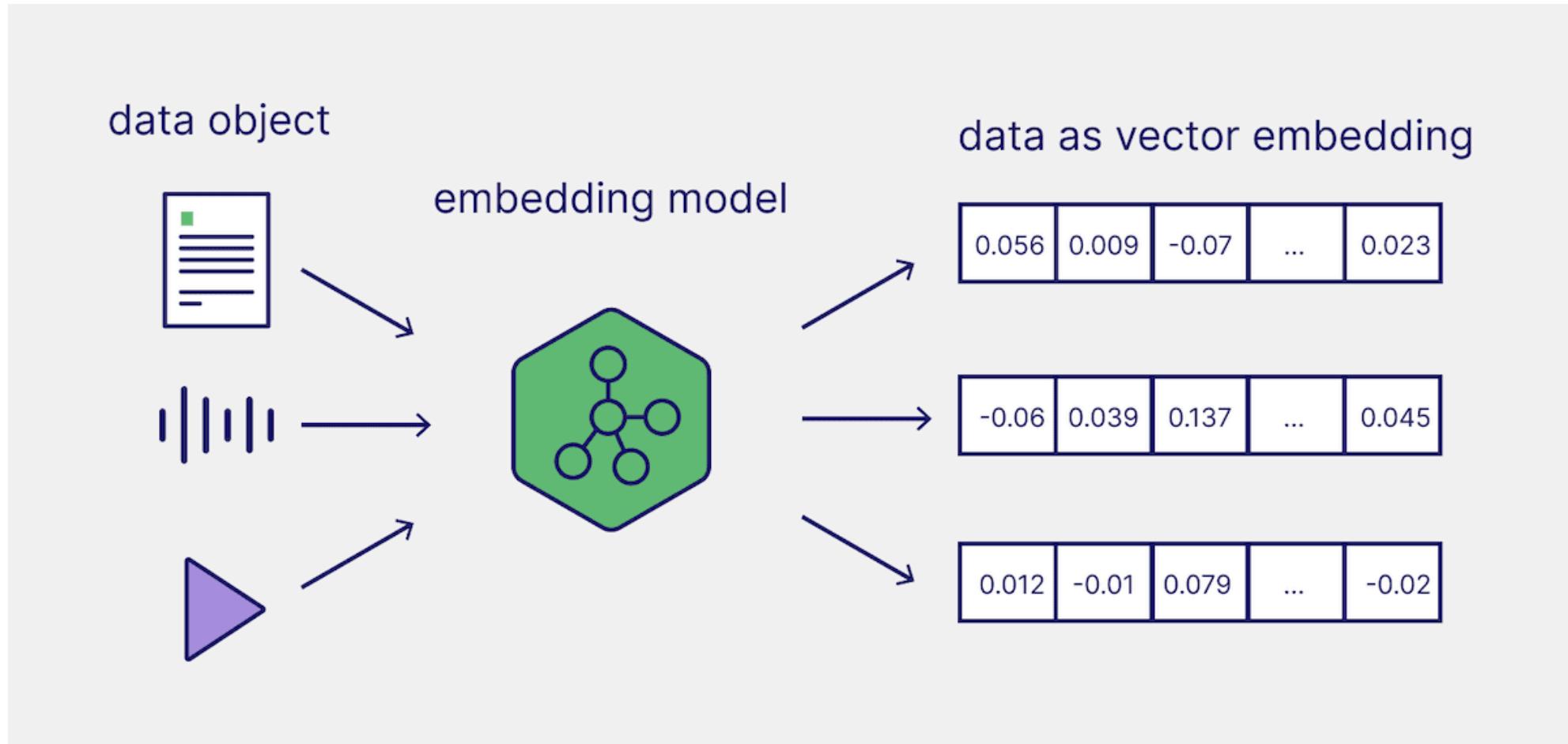


Hình 8: Tìm kiếm hình ảnh tương tự sử dụng Google Lens

# Mục lục

- 1. Giới thiệu**
- 2. Ứng dụng thực tế**
- 3. Biểu diễn dữ liệu**
- 4. Các phép đo độ tương đồng**
- 5. Các kỹ thuật tìm cặp tương đồng**

### 3.1. Vector

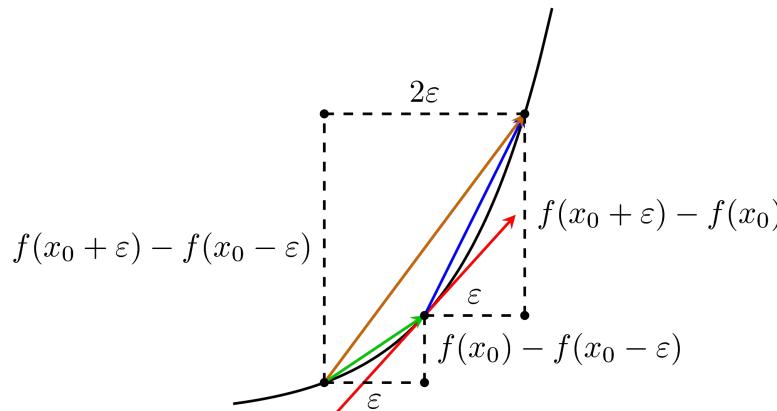


Hình 9: Công dụng của một mô hình embedding

# 3.1. Vector

## KIẾU BIỂU DIỄN PHỐ BIỀN NHẤT

- Các mô hình học máy dựa trên nền tảng của đại số:



Hình 10: Đạo hàm

**a**

$$\begin{pmatrix} c_1 & c_2 \\ c_3 & c_4 \end{pmatrix} = \begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \end{pmatrix} \cdot \begin{pmatrix} b_1 & b_2 \\ b_3 & b_4 \end{pmatrix}$$

**b**

$$m_1 = (a_1 + a_4)(b_1 + b_4)$$
$$m_2 = (a_3 + a_4)b_1$$
$$m_3 = a_1(b_2 - b_4)$$
$$m_4 = a_4(b_3 - b_1)$$
$$m_5 = (a_1 + a_2)b_4$$
$$m_6 = (a_3 - a_1)(b_1 + b_2)$$
$$m_7 = (a_2 - a_4)(b_3 + b_4)$$
$$c_1 = m_1 + m_4 - m_5 + m_7$$
$$c_2 = m_3 + m_5$$
$$c_3 = m_2 + m_4$$
$$c_4 = m_1 - m_2 + m_3 + m_6$$

**c**

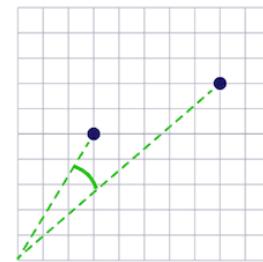
$$\mathbf{u} = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & -1 \end{pmatrix}$$
$$\mathbf{v} = \begin{pmatrix} 1 & 1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & -1 & 0 & 1 & 0 & 1 \end{pmatrix}$$
$$\mathbf{w} = \begin{pmatrix} 1 & 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Hình 11: Nhân ma trận

### 3.1. Vector

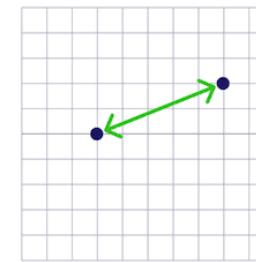
- Có thể đo khoảng cách một cách nhanh chóng và dễ dàng

## Distance Metrics in Vector Search



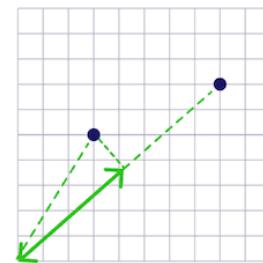
Cosine Distance

$$1 - \frac{A \cdot B}{\|A\| \|B\|}$$



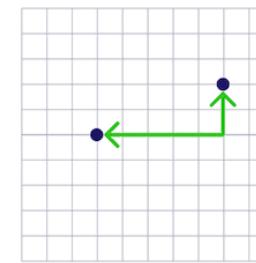
Squared Euclidean  
(L2 Squared)

$$\sum_{i=1}^n (x_i - y_i)^2$$



Dot Product

$$A \cdot B = \sum_{i=1}^n A_i B_i$$



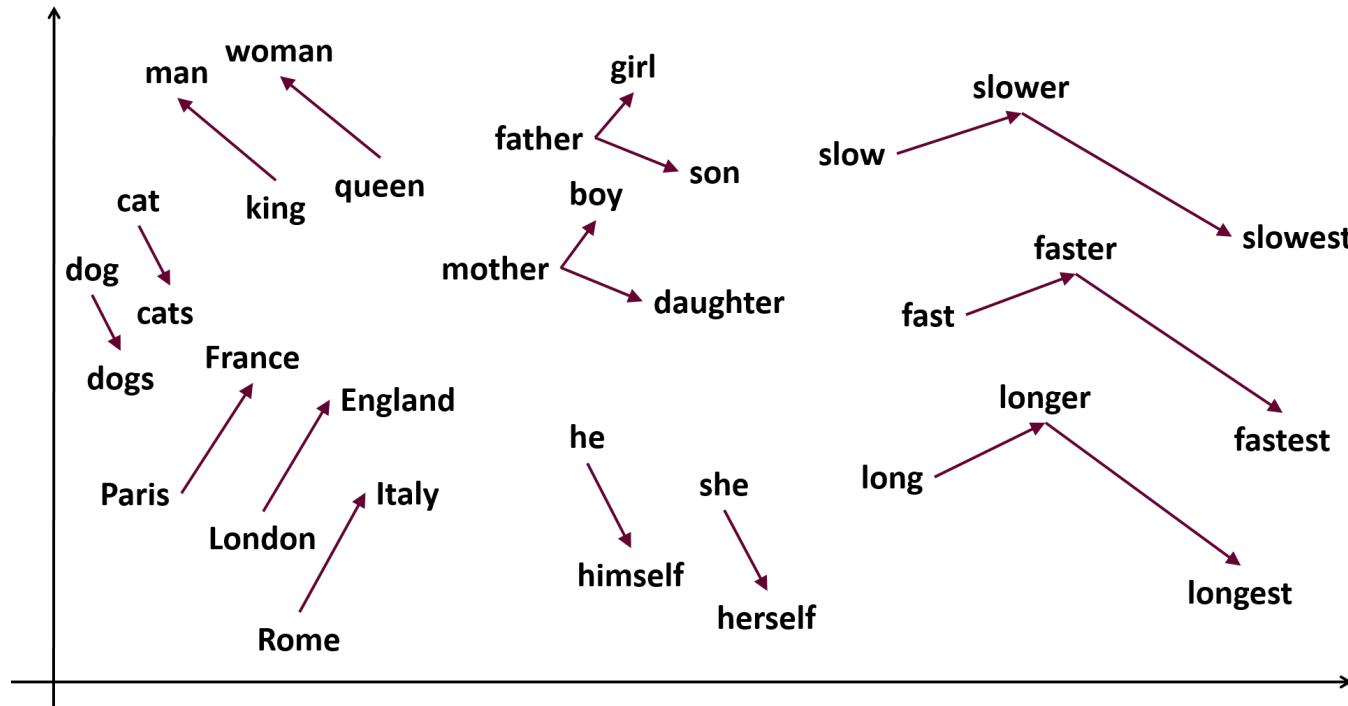
Manhattan (L1)

$$\sum_{i=1}^n |x_i - y_i|$$

Hình 12: Khoảng cách vector trong không gian

### 3.1. Vector

- Dễ cải tiến tốc độ sử dụng GPU
- Tạo ra không gian vector liên tục giúp mô hình có thể tự hiểu ngữ nghĩa



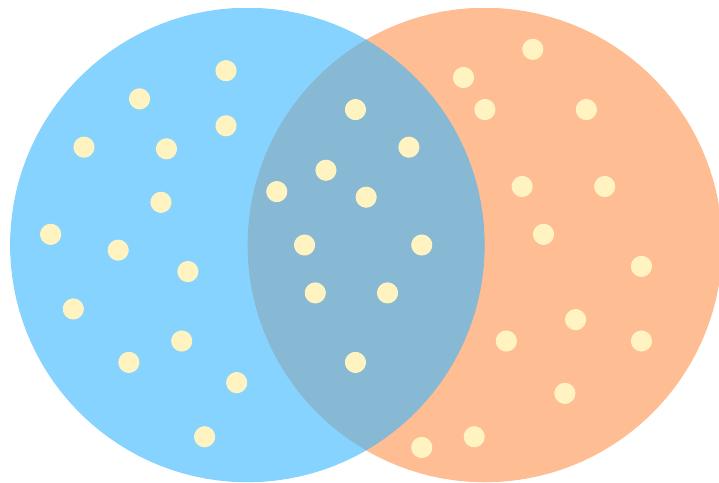
Hình 13: Không gian các từ

## 3.2. Hashing

# Mục lục

1. Giới thiệu
2. Ứng dụng thực tế
3. Biểu diễn dữ liệu
- 4. Các phép đo độ tương đồng**
5. Các kỹ thuật tìm cặp tương đồng

## 4.1. Jaccard Similarity



$$J(A, B) = \frac{A \cap B}{A \cup B}$$

Hình 14: Jaccard Similarity

## 4.2. Phép đo khoảng cách

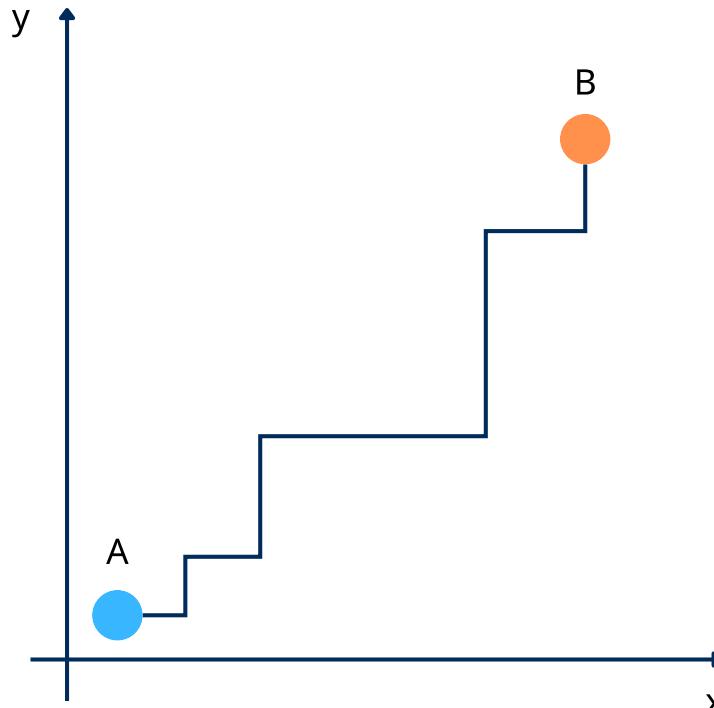
### 4.2.1. $L_p$ norm

Dạng tổng quát:

$$d_p(x, y) = \left( \sum_{i=1}^n |x_i - y_i|^p \right)^{\frac{1}{p}}$$

## 4.2. Phép đo khoảng cách

Với  $p = 1$ , ta có khoảng cách Manhattan:

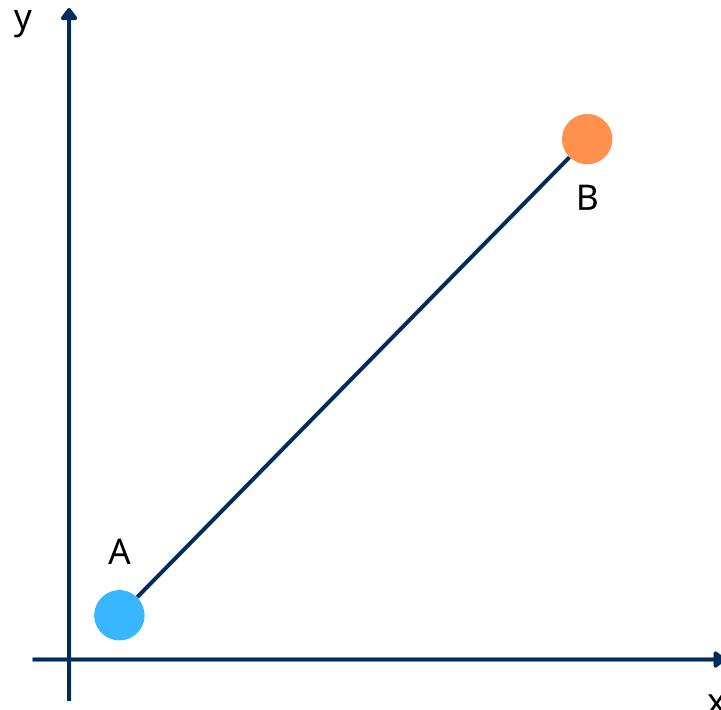


$$d_1(x, y) = \sum_{i=1}^n |x_i - y_i|$$

Hình 15: Manhattan Distance

## 4.2. Phép đo khoảng cách

Với  $p = 2$ , ta có khoảng cách Euclidean:



$$d_2(x, y) = \sqrt{\sum_{i=1}^n |x_i - y_i|^2}$$

Hình 16: Euclidean Distance

## 4.2. Phép đo khoảng cách

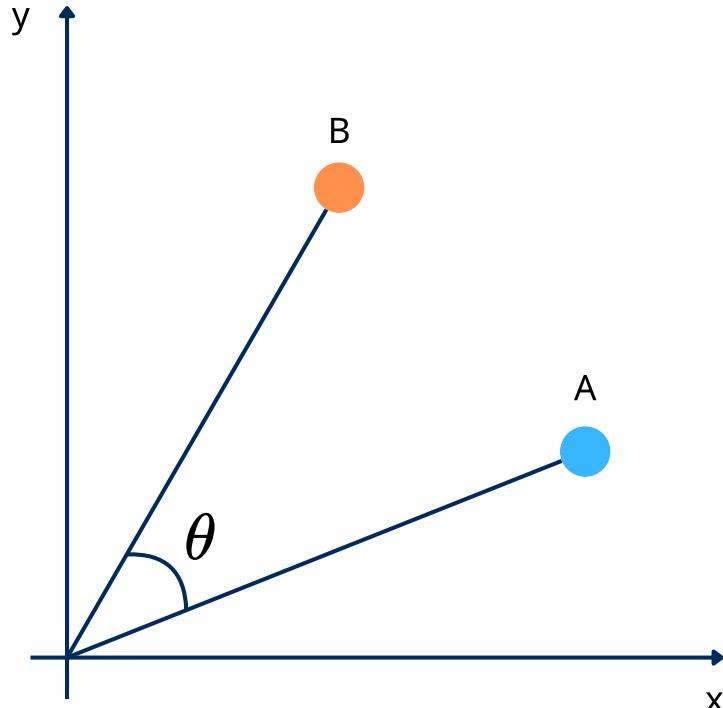
### 4.2.2. Jaccard Distance

$$d_J(A, B) = 1 - J(A, B)$$

Trong đó  $J(A, B)$  là Jaccard Similarity giữa hai tập  $A$  và  $B$ .

## 4.2. Phép đo khoảng cách

### 4.2.3. Cosine Similarity



$$\cos(\theta) = \frac{A \cdot B}{\|A\| * \|B\|}$$

Hình 17: Cosine Similarity

## 4.2. Phép đo khoảng cách

4.2.4. Edit Distance

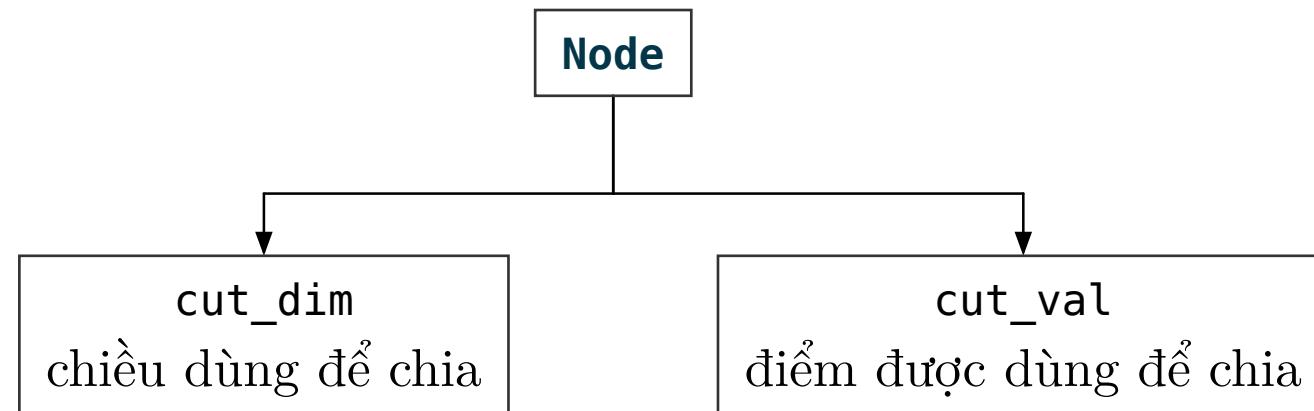
4.2.5. Hamming Distance

## 4.3. So sánh

# Mục lục

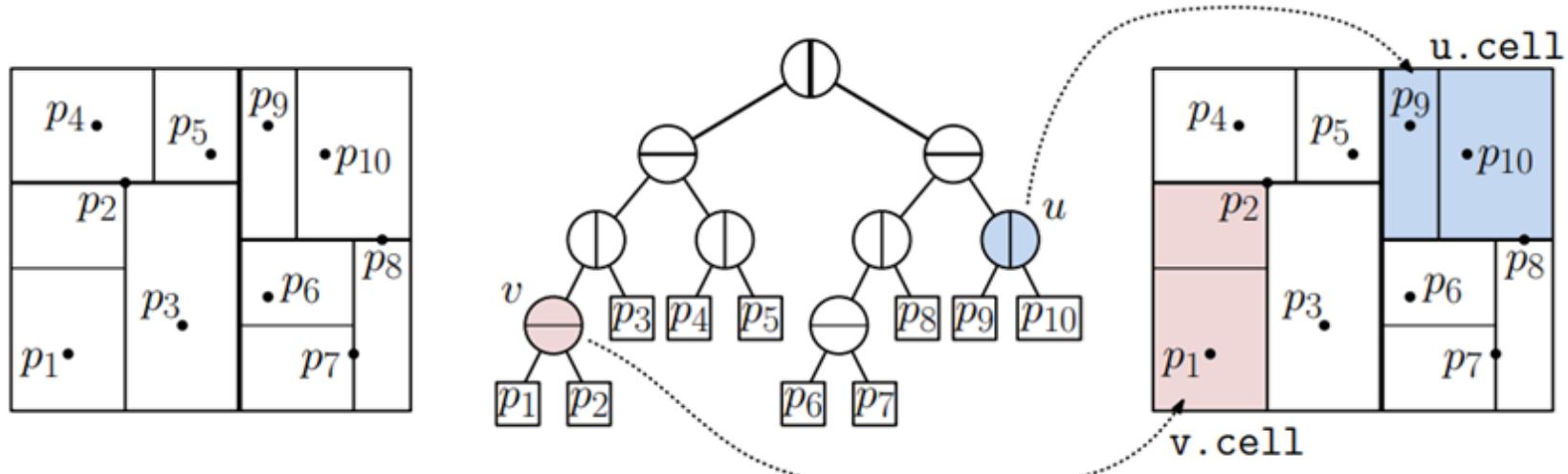
1. Giới thiệu
2. Ứng dụng thực tế
3. Biểu diễn dữ liệu
4. Các phép đo độ tương đồng
5. Các kỹ thuật tìm cặp tương đồng

## 5.1. K-D Tree



Hình 18: Cấu trúc K-d Tree

## 5.1. K-D Tree



Hình 19: Ví dụ cây với một tập điểm 2D

## 5.1. K-D Tree

### Xây dựng cây

---

#### Thuật toán 1: Xây dựng cây K-d Tree

---

```
1 function build( $P$ , depth)
2   if  $P$  is empty then
3     return None
4   if  $|P| = 1$  then
5     return Node(depth mod  $k$ ,  $P[0]$ , None, None)
6   mid  $\leftarrow$  midpoint to partition  $P$  into  $P_0$  and  $P_1$  by depth mod  $k$ 
7   left_node  $\leftarrow$  build( $P_0$ , depth + 1)
8   right_node  $\leftarrow$  build( $P_1$ , depth + 1)
9   return Node(depth mod  $k$ , mid, left_node, right_node)
```

---

Độ phức tạp:

- Giả sử ở mỗi bước đệ quy ta chia đôi tập điểm  $\Rightarrow \mathcal{O}(n \log n)$
- Tìm trung vị:
  - Quicksort:  $\mathcal{O}(n \log^2 n)$
  - Median of medians:  $\mathcal{O}(n \log n)$
  - Duy trì  $k$  danh sách sắp xếp theo từng chiều:  $\mathcal{O}(nk \log n)$

## 5.1. K-D Tree

### Chèn

---

**Thuật toán 2:** Chèn điểm vào cây K-d Tree

---

```
1 function insert( $x$ , node, depth)
2   if node is empty then
3      $\sqsubset$  return Node(depth mod  $k$ ,  $x$ , None, None)
4   if  $x == \text{node.data}$  then
5      $\sqsubset$  return error “duplicate”
6   cd  $\leftarrow$  depth mod  $k$ 
7   if  $x[cd] < \text{node.data}[cd]$  then
8      $\sqsubset$  node.left  $\leftarrow$  insert( $x$ , node.left, depth + 1)
9   else
10     $\sqsubset$  node.right  $\leftarrow$  insert( $x$ , node.right, depth + 1)
11  return node
```

---

Độ phức tạp:

- Ngẫu nhiên:  $\mathcal{O}(\log n)$
- Tối đa:  $\mathcal{O}(n)$

## 5.1. K-D Tree

### Tìm điểm có chiều $d$ nhỏ nhất

---

**Thuật toán 3:** Tìm điểm có chiều  $d$  nhỏ nhất

---

```
1 function findMin(root, d)
2   if root is empty then
3     return None
4   if root.cut_dim == d then
5     if root.left is empty then
6       return root
7     return findMin(root.left, d)
8   else
9     left_min ← findMin(root.left, d)
10    right_min ← findMin(root.right, d)
11    res ← root
12    for candidate in [left_min, right_min] do
13      if candidate != None and candidate.point[d] < res.point[d] then
14        res ← candidate
15    return res
```

---

# 5.1. K-D Tree

## Xóa

---

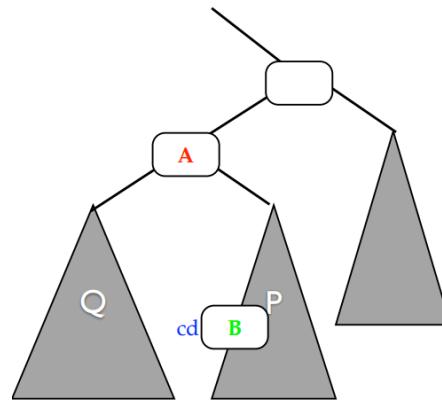
**Thuật toán 4:** Xóa điểm trong cây K-d Tree

---

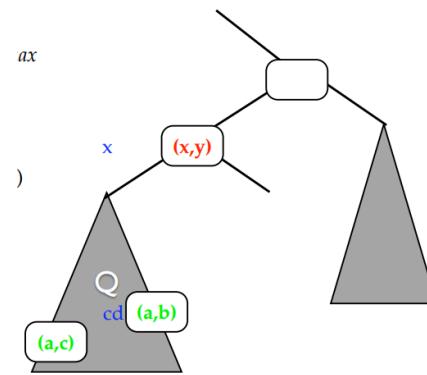
```
1 function delete(root, point, depth)
2   if root is empty then
3     return None
4   if root.point == point then
5     if root.right is not empty then
6       minNode ← findMin(root.right, root.cut_dim)
7       root.point ← minNode.point
8       root.right ← delete(root.right, minNode.point, depth + 1)
9     else if root.left is not empty then
10      minNode ← findMin(root.left, root.cut_dim)
11      root.point ← minNode.point
12      root.right ← delete(root.left, minNode.point, depth + 1)
13      root.left ← None
14    else
15      return None
16    return root
17    axis ← depth mod k
18    if point[axis] < root.point[axis] then
19      root.left ← delete(root.left, point, depth + 1)
20    else
21      root.right ← delete(root.right, point, depth + 1)
22    return root
```

---

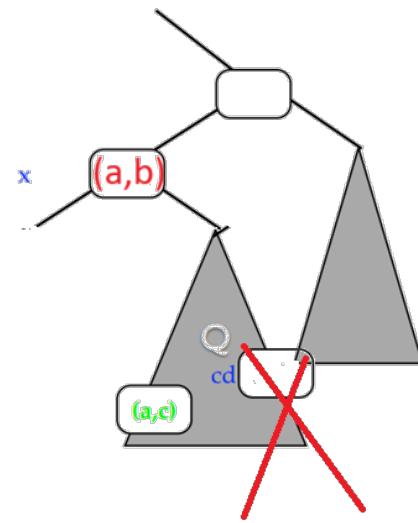
## 5.1. K-D Tree



Hình 20: Tồn tại con bên phải



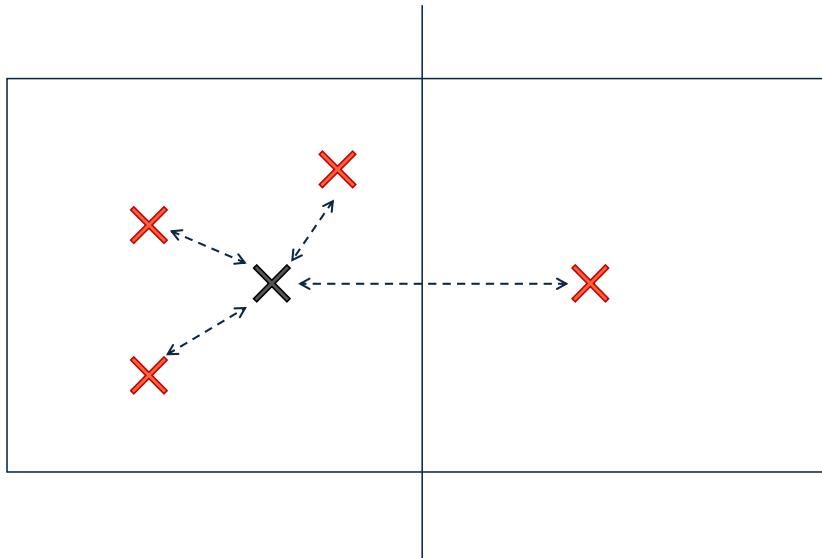
,  
Hình 21: Không tồn tại  
con bên phải



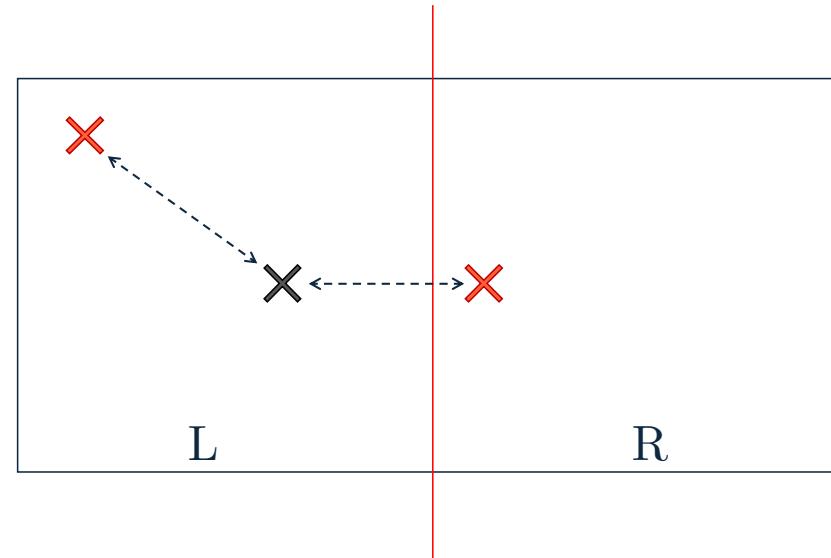
Hình 22: Không tồn tại  
con bên phải

## 5.1. K-D Tree

Tìm  $k$  điểm gần nhất.



Hình 23: Cắt tia giúp giảm số lượng điểm duyệt



Hình 24: Cắt tia không làm giảm số lượng điểm cần duyệt

# 5.1. K-D Tree

---

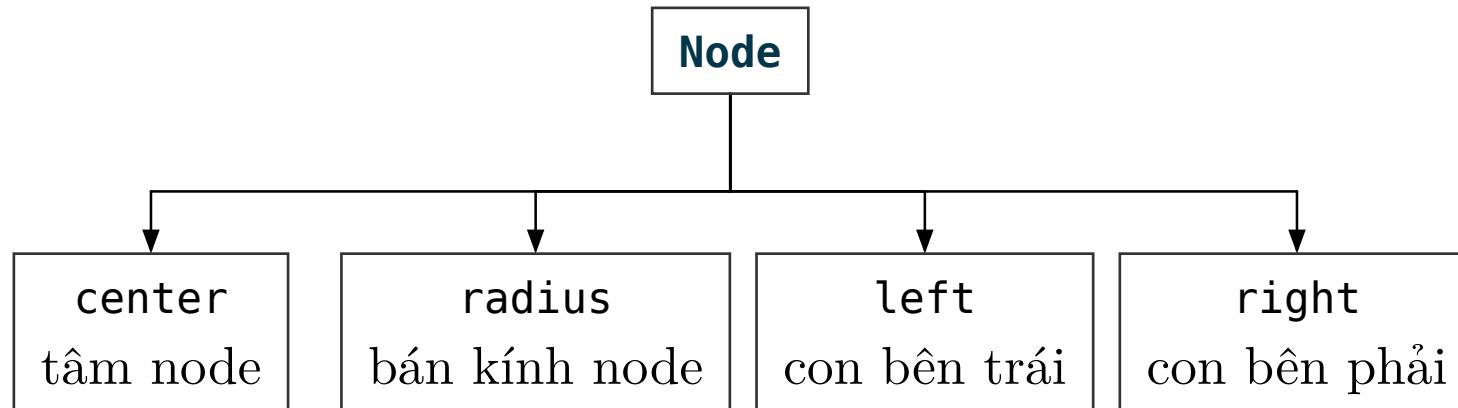
**Thuật toán 5:** Tìm  $k$  điểm gần nhất

---

```
1 function kNearestNeighbor(root, depth, point, k)
2     if root is empty then
3         return
4     cut_dim ← depth mod  $d$ 
5     if point[cut_dim] < root.cut_val[cut_dim] then
6         if root.left is not empty then
7             ↘ kNearestNeighbor(root.left, depth + 1, point, k)
8             estimated_dist ← (point[cut_dim] - root.cut_val[cut_dim])2
9             if estimated_dist ≤ pq.maxDist() or len(pq) < k then
10                if root.right is not empty then
11                    ↘ kNearestNeighbor(root.right, depth + 1, point, k)
12                    pq.add(root.cut_val)
13                else if point[cut_dim] > root.cut_val[cut_dim] then
14                    if root.right is not empty then
15                        ↘ kNearestNeighbor(root.right, depth + 1, point, k)
16                        estimated_dist ← (point[cut_dim] - root.cut_val[cut_dim])2
17                        if estimated_dist ≤ pq.maxDist() or len(pq) < k then
18                            if root.left is not empty then
19                                ↘ kNearestNeighbor(root.left, depth + 1, point, k)
20                                pq.add(root.cut_val)
21                            else
22                                if root.left is not empty then
23                                    ↘ kNearestNeighbor(root.left, depth + 1, point, k)
24                                if root.right is not empty then
25                                    ↘ kNearestNeighbor(root.right, depth + 1, point, k)
26                                pq.add(root.cut_val)
```

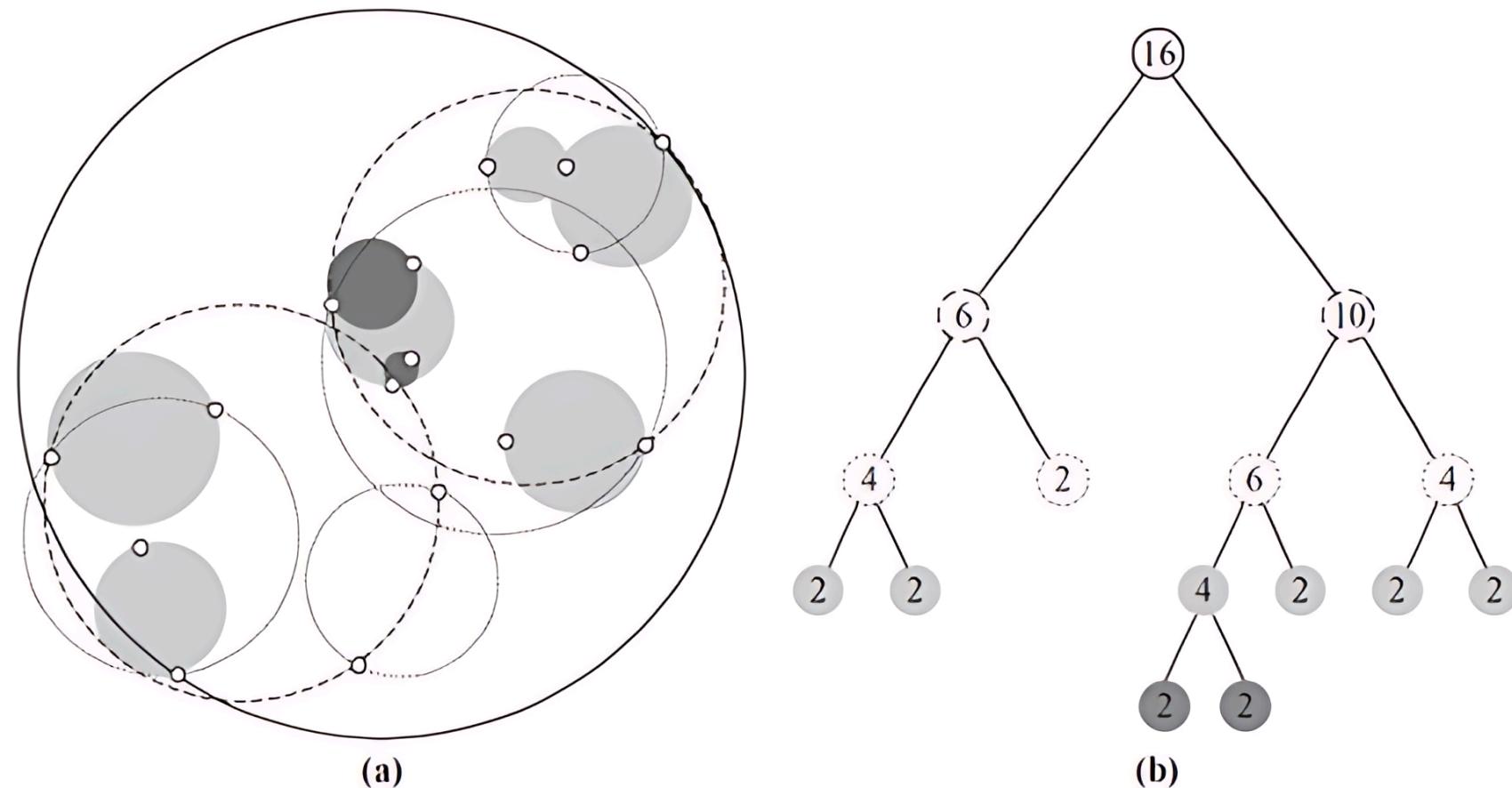
---

## 5.2. Ball Tree



Hình 25: Cấu trúc Ball Tree

## 5.2. Ball Tree



Hình 26: Minh họa Ball Tree

## 5.2. Ball Tree

### Xây dựng cây

---

**Thuật toán 6:** Xây dựng cây Ball Tree

---

```
1 function CONSTRUCT_BALLTREE( $D$ , max_leaf_size)
2    $N = |D|$ 
3   center  $\leftarrow$  CENTROID( $D$ )
4   radius  $\leftarrow \max_{x \in D} \|x - \text{center}\|$ 
5   node  $\leftarrow$  new Node(center=center, radius=radius)
6   if  $N \leq \text{max\_leaf\_size}$  then
7     node.points  $\leftarrow D$ 
8     node.is_leaf  $\leftarrow$  True
9     return node
10  else
11     $p_L \leftarrow \arg \max_{x \in D} \|x - \text{center}\|$ 
12     $p_R \leftarrow \arg \max_{x \in D} \|x - p_L\|$ 
13     $D_{\text{left}} \leftarrow \{x \in D \mid \|x - p_L\| \leq \|x - p_R\|\}$ 
14     $D_{\text{right}} \leftarrow D \setminus D_{\text{left}}$ 
15    node.left  $\leftarrow$  CONSTRUCT_BALLTREE( $D_{\text{left}}$ , max_leaf_size)
16    node.right  $\leftarrow$  CONSTRUCT_BALLTREE( $D_{\text{right}}$ , max_leaf_size)
17    return node
```

---

Độ phức tạp:

- Trung bình:  $\mathcal{O}(Nd \log N)$
- Xấu nhất :  $\mathcal{O}(N^2d)$  khi split rất lệch

## 5.2. Ball Tree

### Chèn

---

**Thuật toán 7:** Chèn điểm vào Ball Tree

---

```
1 function INSERT(node, x)
2     if node.is_leaf then
3         node.points.append(x)
4         if |node.points| > max_leaf_size then
5             ↘ rebuild(node)
6     else
7         if dist(x, node.left.center) < dist(x, node.right.center)
8             then
9                 ↘ INSERT(node.left, x)
10            else
11                ↘ INSERT(node.right, x)
12    update(node.center, node.radius)
```

---

Độ phức tạp:

- Trung bình:  $\mathcal{O}(d \log N)$
- Xấu nhất :  $\mathcal{O}(dN)$

## 5.2. Ball Tree

### Xóa

---

#### Thuật toán 8: Xóa điểm khỏi Ball Tree

---

```
1 function DELETE(node, x)
2   if node.is_leaf then
3     ↘ remove(x, node.points)
4   else
5     if dist(x, node.left.center) < dist(x, node.right.center) then
6       ↘ DELETE(node.left, x)
7     else
8       ↘ DELETE(node.right, x)
9       ↘ update(node.center, node.radius)
```

---

Độ phức tạp:

- Trung bình:  $\mathcal{O}(d \log N)$
- Xấu nhất :  $\mathcal{O}(dN)$

## 5.2. Ball Tree

### Tìm $k$ điểm gần nhất

Thuật toán 9: Tìm  $k$  điểm gần nhất

```
1 function BALLTREE_KNN(root, q, k)
2     heap ← empty max-heap
3     function SEARCH(node)
4         if node is leaf then
5             for p in node.points do
6                 d ← dist(q, p)
7                 if heap.size < k then
8                     heap.push((d, p))
9                 else if d < heap.max_key then
10                    heap.replace_max((d, p))
11        return
12        dL ← max(0, dist(q, node.left.center) - node.left.radius)
13        dR ← max(0, dist(q, node.right.center) - node.right.radius)
14        if dL < dR then
15            if heap.size < k or dL ≤ heap.max_key then
16                SEARCH(node.left)
17            if heap.size < k or dR ≤ heap.max_key then
18                SEARCH(node.right)
19        else
20            if heap.size < k or dR ≤ heap.max_key then
21                SEARCH(node.right)
22            if heap.size < k or dL ≤ heap.max_key then
23                SEARCH(node.left)
24        SEARCH(root)
25    return heap.items sorted asc by distance
```

Độ phức tạp:

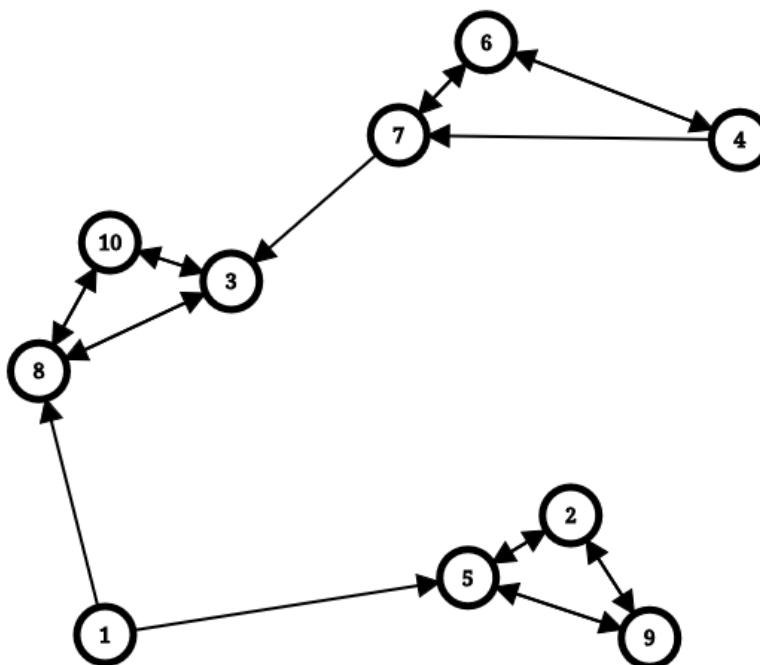
- Trung bình:  $\mathcal{O}(d \log N)$ 
  - $d$  để tính khoảng cách từ truy vấn đến các node con trong cây
  - $\log N$  số tầng trung bình của cây
- Xấu nhất :  $\mathcal{O}(dN)$

## 5.3. LSH

## 5.4. K-Nearest-Neighbor Graph

### 5.4.1. Định nghĩa

- Mỗi đỉnh có cạnh nối tới  $k$  đỉnh gần nhất



Hình 27: Ví dụ đồ thị có  $k = 3$ , khoảng cách Euclid

## 5.4. K-Nearest-Neighbor Graph

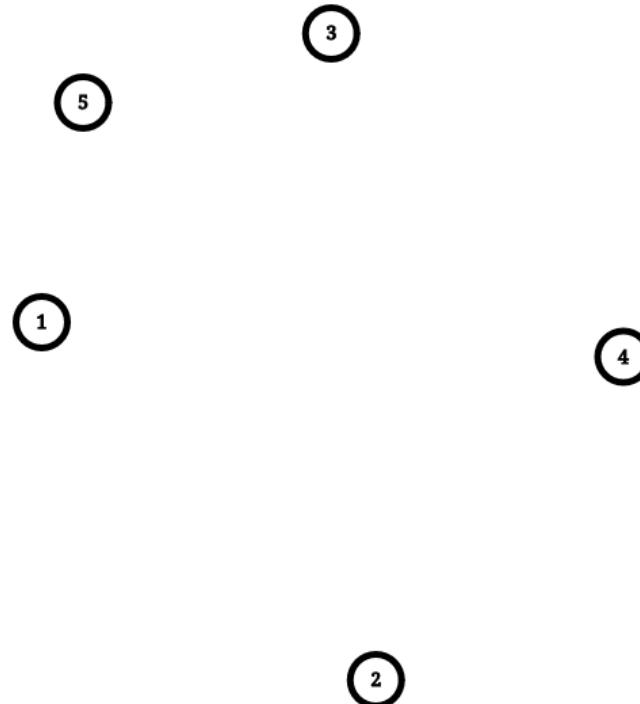
### 5.4.2. Xây dựng

- Brute-force: Tính khoảng cách giữa mọi cặp đỉnh
- Độ phức tạp:  $\mathcal{O}(n^2d)$

## 5.4. K-Nearest-Neighbor Graph

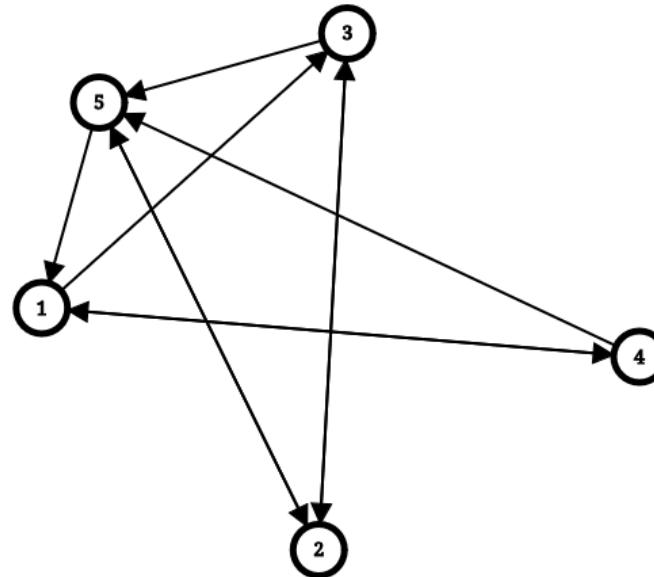
- *Efficient K-Nearest Neighbor Graph Construction for Generic Similarity Measures*
- Hàng xóm của hàng xóm khả năng cao cũng là hàng xóm
- Bắt đầu với đồ thị ngẫu nhiên, cải thiện dần

## 5.4. K-Nearest-Neighbor Graph



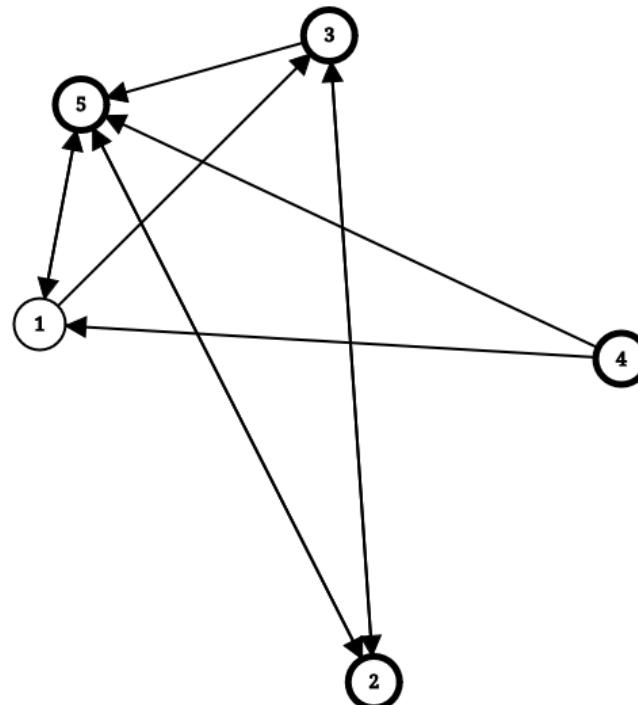
Hình 28: Tập đỉnh cần tạo đồ thị

## 5.4. K-Nearest-Neighbor Graph



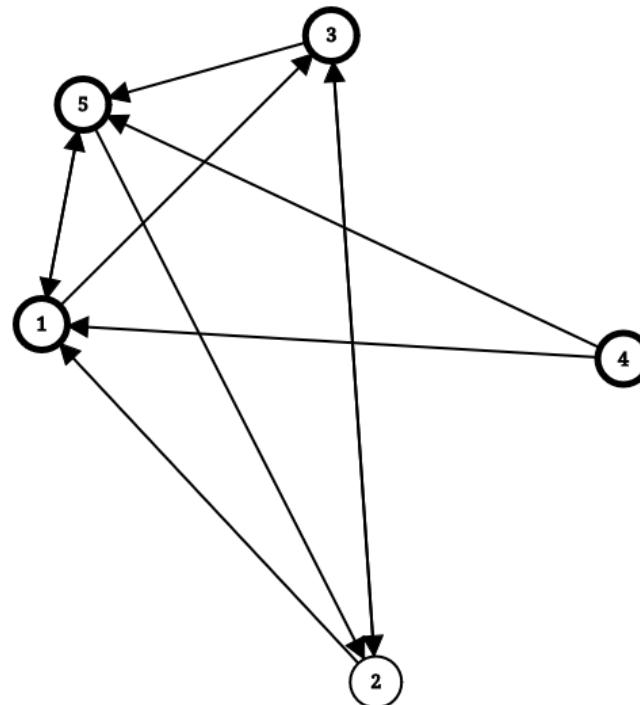
Hình 29: Đồ thị ban đầu được khởi tạo ngẫu nhiên

## 5.4. K-Nearest-Neighbor Graph



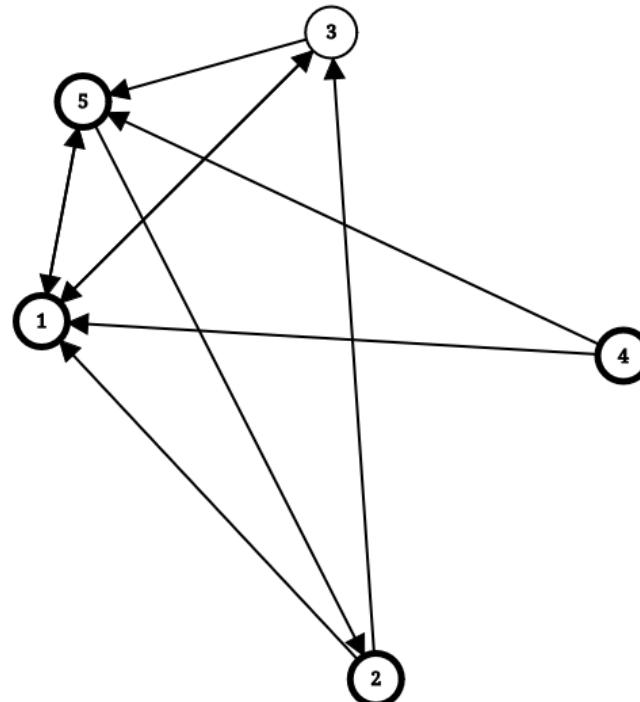
Hình 30: Cải thiện hàng xóm đỉnh 1

## 5.4. K-Nearest-Neighbor Graph



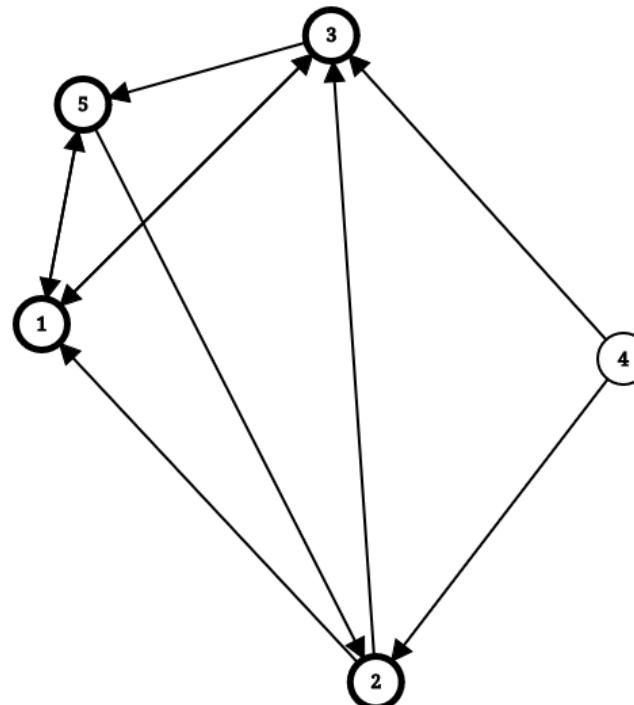
Hình 31: Cải thiện hàng xóm đỉnh 2

## 5.4. K-Nearest-Neighbor Graph



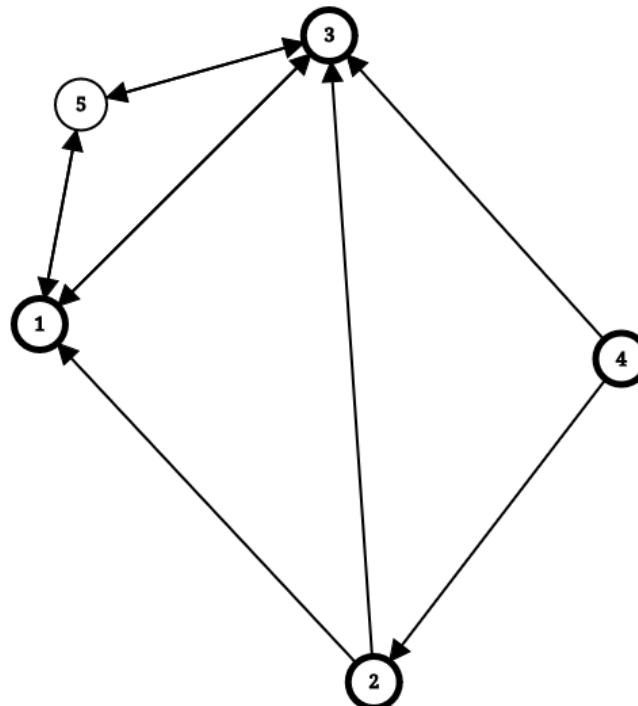
Hình 32: Cải thiện hàng xóm đỉnh 3

## 5.4. K-Nearest-Neighbor Graph



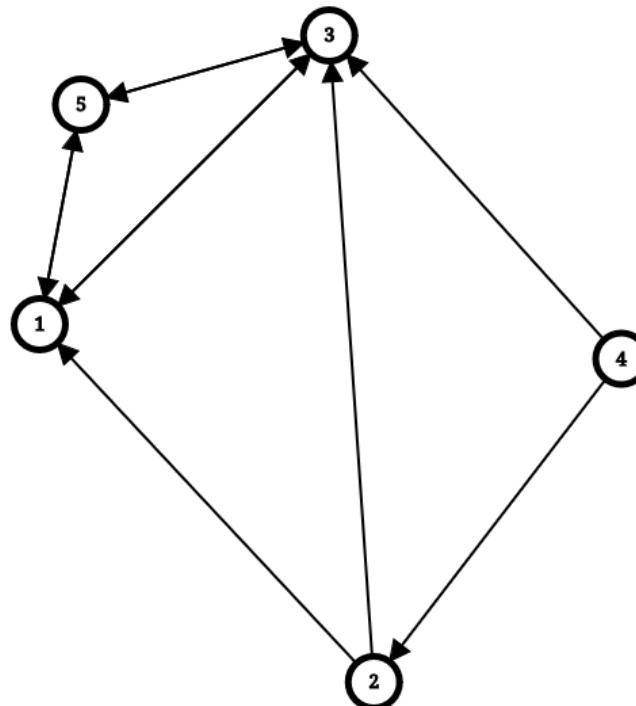
Hình 33: Cải thiện hàng xóm đỉnh 4

## 5.4. K-Nearest-Neighbor Graph



Hình 34: Cải thiện hàng xóm đỉnh 5

## 5.4. K-Nearest-Neighbor Graph



Hình 35: Không còn cải thiện thêm được nữa

## 5.4. K-Nearest-Neighbor Graph

---

### Thuật toán 10: Xây dựng KNNG với NNDescent

---

```
1 function NNDescent(V)
2   for u in V do
3     ↘ randomly pick k other nodes
4   while number of updates < threshold do
5     for u in V do
6       for v in neighbors(u) do
7         for v' in neighbors(v) do
8           ↘ compute  $d(u, v')$ 
9   pick out k closest nodes to u, update u's neighbor list
```

---

Dộ phức tạp thuật toán là  $\mathcal{O}(mnk^2d)$

- $m$  lần lặp
- Mỗi lần duyệt mất  $nk^2$
- Tính khoảng cách mất  $d$

Ưu điểm:

- Hoạt động với mọi hàm khoảng cách
- Sử dụng thêm ít bộ nhớ
- Độ chính xác cao
- Dễ cài đặt

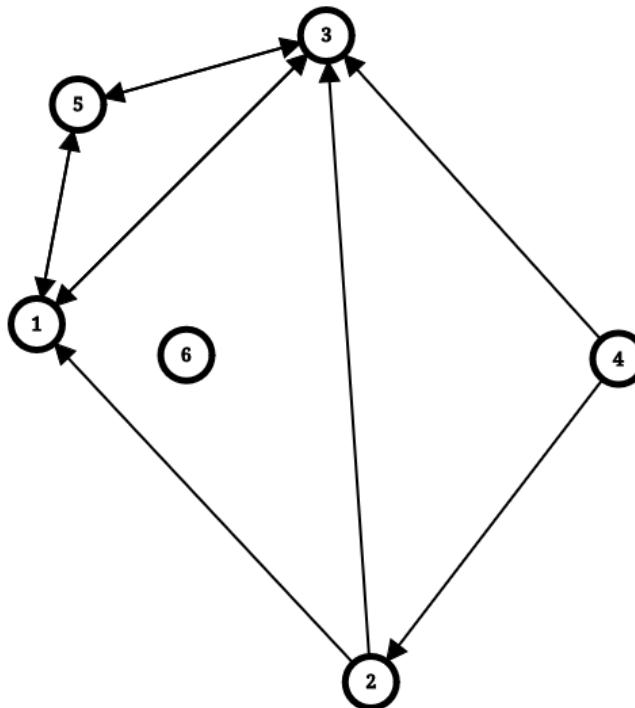
## 5.4. K-Nearest-Neighbor Graph

### 5.4.3. Truy vấn

- Heuristic/meta-heuristic
- Di chuyển theo hướng có các đỉnh gần đỉnh truy vấn  $q$  nhất

## 5.4. K-Nearest-Neighbor Graph

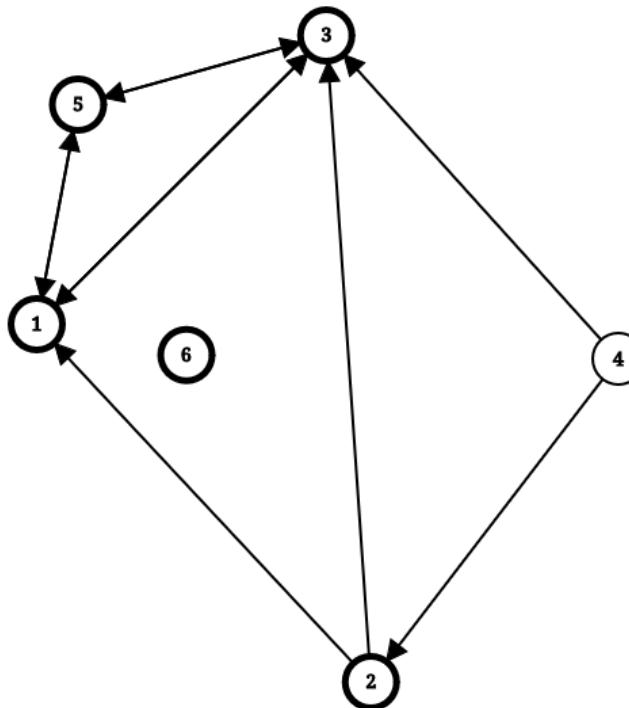
[4]



Hình 36: Chọn ngẫu nhiên đỉnh số 4 để bắt đầu

## 5.4. K-Nearest-Neighbor Graph

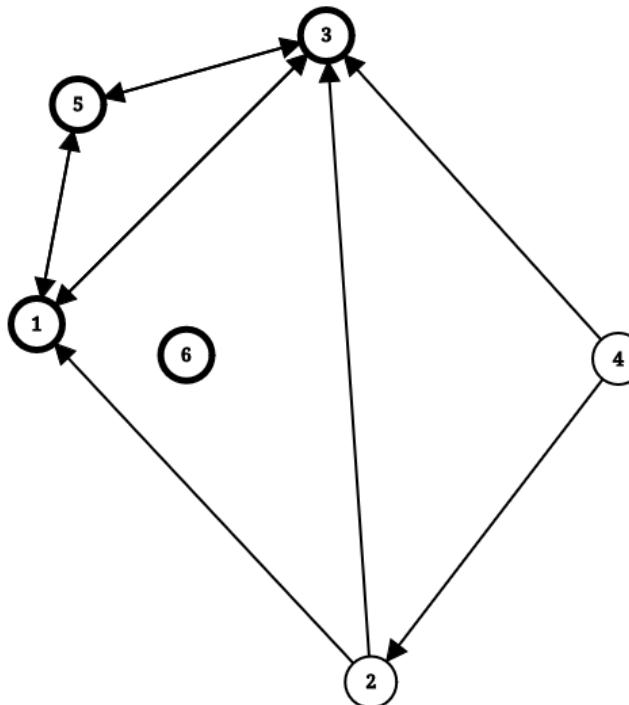
[4, 3, 2]



Hình 37: Thăm đỉnh 4, xét hàng xóm là 2 và 3

## 5.4. K-Nearest-Neighbor Graph

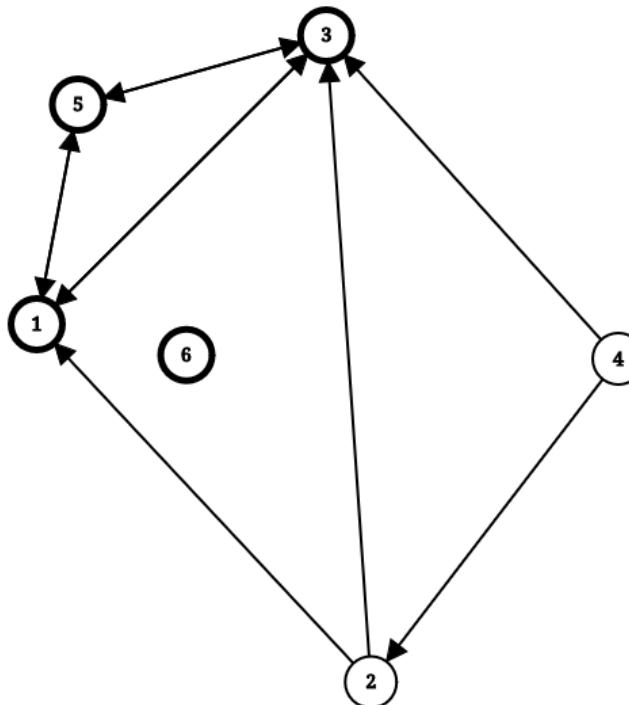
[4, 3, 2, 1]



Hình 38: Thăm đỉnh 2, xét hàng xóm là 1 và 3

## 5.4. K-Nearest-Neighbor Graph

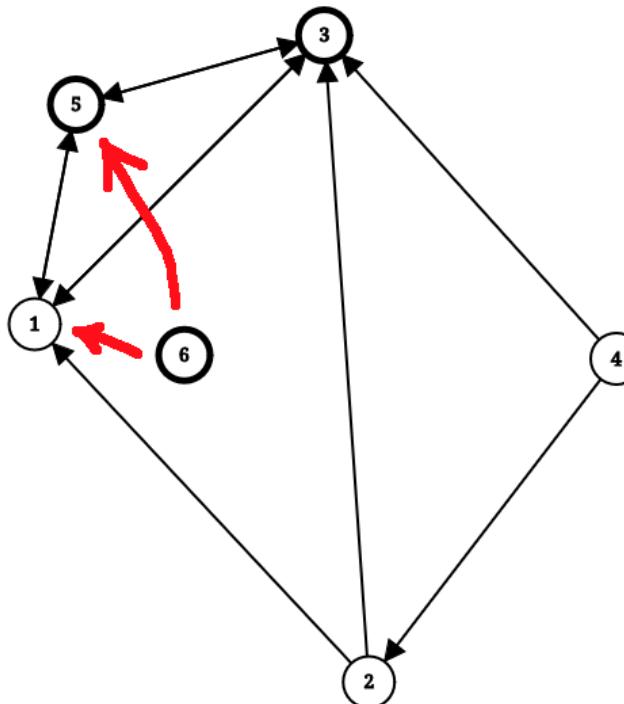
[4, 3, 2, 1]



Hình 39: Thăm đỉnh 2, xét hàng xóm là 1 và 3

## 5.4. K-Nearest-Neighbor Graph

[4, 3, 2, 1, 5]



Hình 40: Thăm đỉnh 1, xét hàng xóm là 3 và 5

## 5.4. K-Nearest-Neighbor Graph

---

### Thuật toán 11: Tìm kiếm trên KNNG

---

```
1 function KNNQuery(G, q, k)
2     variable visited = generateStartingNodes(G)
3     variable min_heap((u, v) -> compare(d(u, q), d(v, q)))
4     = visited
5     while stopping conditions not met
6         variable u = min_heap.pop()
7         for v in neighbors(u)
8             if v not in visited
9                 visited.add(v)
10                min_heap.add(v)
```

---

Độ phức tạp thuật toán là  $\mathcal{O}(m(\log(m) + k_{\text{graph}}(d + \log(m))))$

- $m$  lần lặp
- Thêm 1 phần tử vào heap mất  $\log(m)$
- Mỗi đỉnh đồ thị có  $k_{\text{graph}}$  hàng xóm

Thực tế nhanh hơn brute-force nhiều

## 5.4. K-Nearest-Neighbor Graph

### 5.4.4. Cập nhật

- Không có hỗ trợ tốt thao tác cập nhật
- Cập nhật cục bộ
- Tái xây dựng sau 1 khoảng thời gian