## **Ví dụ**

2 1

hackerearth 10

hackerrank 9

hacker

* Các kết quả tìm kiếm cho chữ hacker là chữ hackerearth và chữ hackerrank. Chữ hackerearth có con số ứng với priority 10 là lớn nhất → Output: 10

## **Hướng giải 1:**

* Lưu độ ưu tiên của từ ở node cuối cùng của từ đó. Lúc này, để tìm từ có tiền tố X và độ ưu tiên lớn nhất, mình phải duyệt qua tất cả từ có tiền tố X. → Độ phức tạp: O(n \* string\_length + **q \* string\_length \* n**).

## **Hướng giải 2:**

* Với mỗi node, lưu độ ưu tiên lớn nhất của từ có chứa node đó. Lúc này, để tìm từ có tiền tố X và độ ưu tiên lớn nhất, mình chỉ cần duyệt đến node cuối cùng của tiền tố X đó và xuất ra độ ưu tiên lớn nhất ứng với node này. → Độ phức tạp: O(n \* string\_length + q \* string\_length).

## **Giải thuật:**

B1: Đọc vào các từ trong database. Khởi tạo cây Trie rỗng.

B2: Thêm n từ vào cây Trie, lưu độ ưu tiên ở tất cả các node mà ta đã đi qua trong quá trình thêm từ đó vào cây.

B3: Đọc vào các truy vấn. Với mỗi truy vấn, duyệt đến node của cuối cùng và xuất ra value của node đó.

## **Mã giả:**

class Node:

Node child [] \* 26

value = -1

insert(root, s, priority):

cur = root

for char in s:

index = char - ‘a’

if cur.child[index] == null:

cur.child = new Node()

cur.value = max(cur.value, priority)

cur = cur.child[index]

getMaxPriority(root, s):

cur = root

for char in s:

index = char - ‘a’

if cur.child[index] == null:

return -1

cur = cur.child[index]

return cur.value

main():

read(N)

root = new Node()

for i = 0 to N - 1:

read(s, priority)

insert(root, s, priority)

read(Q)

for i = 0 to Q - 1:

read(s)

print(getMaxPriority(s))

Độ phức tạp: O(n \* string\_length + q \* string\_length)