**Problem: Website Similarity**

**Scenario**

You work for a company that tracks user visits to various websites. Your team wants to analyze user behaviour to identify pairs of websites that have the most similar user bases. Given a list of tuples representing user visits to websites, you need to develop a program that identifies the top k pairs of websites with the greatest similarity in their user bases.

**Problem Statement**

Given a list of (website, user) pairs that represent users visiting websites, come up with a program that identifies the top k pairs of websites with the greatest similarity.

**Similarity Metric**

A reasonable similarity metric for this problem is the Jaccard similarity index, which measures the similarity between finite sample sets. The Jaccard similarity index is defined as the size of the intersection divided by the size of the union of the sample sets.

**Input Format**

* The first line contains an integer k, representing the number of top pairs to return.
* The second line contains an integer n, representing the number of (website, user) pairs.
* The next n lines each contain a tuple with a string website and an integer user, representing a user visiting a website.

**Constraints**

* 1 <= k <= n \* (n - 1) / 2
* 1 <= n <= 10000
* 1 <= |website| <= 100
* 1 <= user <= 10000

**Output Format**

* Print the top k pairs of websites with the greatest similarity in descending order of similarity. Each pair should be represented as a tuple of two website names.

**Sample Input**

1

17

a 1

a 3

a 5

b 2

b 6

c 1

c 2

c 3

c 4

c 5

d 4

d 5

d 6

d 7

e 1

e 3

e 5

e 6

**Sample Output**

[('a', 'e')]

**Explanation**

The most similar websites are a and e based on their user bases.

**Solution**

Here's the solution in Python:

python

from collections import defaultdict

from itertools import combinations

def jaccard\_similarity(set1, set2):

intersection = len(set1 & set2)

union = len(set1 | set2)

return intersection / union

def top\_k\_similar\_websites(k, visits):

# Build a dictionary where the key is the website and the value is a set of users

website\_users = defaultdict(set)

for website, user in visits:

website\_users[website].add(user)

# Calculate similarity for all pairs of websites

similarities = []

websites = list(website\_users.keys())

for w1, w2 in combinations(websites, 2):

sim = jaccard\_similarity(website\_users[w1], website\_users[w2])

similarities.append((sim, w1, w2))

# Sort the similarities in descending order and return the top k pairs

similarities.sort(reverse=True, key=lambda x: x[0])

return [(w1, w2) for \_, w1, w2 in similarities[:k]]

# Input reading

k = int(input())

n = int(input())

visits = []

for \_ in range(n):

website, user = input().split()

visits.append((website, int(user)))

# Output

result = top\_k\_similar\_websites(k, visits)

print(result)

**Explanation of the Solution**

1. **Data Collection**: The solution first builds a dictionary website\_users where the key is the website and the value is a set of users who visited that website.
2. **Similarity Calculation**: It then calculates the Jaccard similarity for all pairs of websites using combinations from the itertools module.
3. **Sorting and Selecting Top k**: The similarities are sorted in descending order, and the top k pairs are returned.
4. **Input and Output Handling**: The solution reads the input, processes it, and prints the result.

**Additional Test Cases**

**Test Case 1**

**Input:**

2

8

a 1

a 2

b 1

b 3

c 2

c 3

d 1

d 2

**Output:**

[('a', 'd'), ('a', 'b')]

**Test Case 2**

**Input:**

3

9

x 1

x 2

x 3

y 2

y 3

y 4

z 3

z 4

z 5

**Output:**

[('x', 'y'), ('y', 'z'), ('x', 'z')]

**Test Case 3**

**Input:**

3

9

x 1

x 2

x 3

y 2

y 3

y 4

y 3

z 4

z 5

**Output:**

[('x', 'y'), ('x', 'z'), ('y', 'z')]

**Test Case 4**

**Input:**

2

8

x 1

x 2

x 3

y 2

y 3

y 4

y 3

z 4

**Output:**

[('x', 'y'), ('y', 'z')]

**Test Case 5**

**Input:**

4

14

x 1

x 2

x 3

y 2

y 3

y 4

y 3

z 4

z 5

z 10

m 2

m 5

n 6

n 1

**Output:**

[('x', 'y'), ('x', 'm'), ('x', 'n'), ('y', 'm')]