Describe how you would calculate the similarity among each of the 3 combination pairs of Product A, B and C; Which ones are most similar: A-B or A-C or B-C?

Ans.

- From the product we can assume the size and colors are irrelevant when calculating similarities, for same product it comes with different sizes and color.
- We can convert category ids and keyword ids as boolean columns for example if the universal set of categories are 0, 1, 2, 4, 5 and one product belongs to category 1, 2 and 3 then as feature we have category_1, category_2, and category_3 equals 1 and category_4, category_5 equals 0
- Similar to categories and keywords, we can convert types and vendors to boolean column as well
- Now we convert all the products into a matrix where row corresponds to a feature and each columns corresponds to a products
- Finally we have a sparse matrix containing ones and zeroes. Where each column corresponds a product. we can compare their Jaccard similarity to find how different they are
- Similarity of A-B 0.3, Similarity of A-C 0.4, Similarity of B-C 0.7
- The code for the calculation is given below

In [1]:
```
item_a = {
    'name': 'a',
    'type': 'top',
    'vendor': 'addidas',
    'color': 'blue',
    'size': 'M',
    'cat_id': [1,2,3,4,5],
    'keyword_id': [10, 22, 30, 31, 34, 55, 57, 120, 345, 453, 456, 665
]
}

item_b = {
    'name': 'b',
    'type': 'bottom',
    'vendor': 'addidas',
    'color': 'blue',
    'size': 'L',
    'cat_id': [1,2,5,7],
    'keyword_id': [22, 31, 77, 222, 234, 543, 665, 976, 987, 1000]
}

item_c = {
    'name': 'c',
    'type': 'top',
    'vendor': 'nike',
    'color': 'blue',
    'size': 'M',
    'cat_id': [1,2,4,5,10],
    'keyword_id': [10, 22, 31, 77, 120, 222, 234, 543, 665, 976, 987,
1000]
}

items = [item_a, item_b, item_c]
items
```

Out[1]:
```
[{'name': 'a',
  'type': 'top',
  'vendor': 'addidas',
  'color': 'blue',
  'size': 'M',
  'cat_id': [1, 2, 3, 4, 5],
  'keyword_id': [10, 22, 30, 31, 34, 55, 57, 120, 345, 453, 456, 665
]},
 {'name': 'b',
  'type': 'bottom',
  'vendor': 'addidas',
  'color': 'blue',
  'size': 'L',
  'cat_id': [1, 2, 5, 7],
  'keyword_id': [22, 31, 77, 222, 234, 543, 665, 976, 987, 1000]},
 {'name': 'c',
  'type': 'top',
  'vendor': 'nike',
  'color': 'blue',
  'size': 'M',
  'cat_id': [1, 2, 4, 5, 10],
  'keyword_id': [10, 22, 31, 77, 120, 222, 234, 543, 665, 976, 987,
1000]}]
```

In [2]:
```
categories = set()
types = set()
keywords = set()

for item in items:
    categories = categories.union(set(item['cat_id']))
    keywords = keywords.union(set(item['keyword_id']))

def create_dummy_column(unionset, column_name, new_col_prefix, item):
    ones = set(item[column_name])
    zeros = unionset - ones
    for column in ones:
        item[new_col_prefix + str(column)] = 1
    for column in zeros:
        item[new_col_prefix + str(column)] = 0
    del item[column_name]
    return item

for item in items:
    item = create_dummy_column(categories, 'cat_id', 'cat_', item)
    item = create_dummy_column(keywords, 'keyword_id', 'keyword_', ite
m)

items
```

Out[2]:
```
[{'name': 'a',
```

```
[{'name': 'a',
  'type': 'top',
  'vendor': 'addidas',
  'color': 'blue',
  'size': 'M',
  'cat_1': 1,
  'cat_2': 1,
  'cat_3': 1,
  'cat_4': 1,
  'cat_5': 1,
  'cat_10': 0,
  'cat_7': 0,
  'keyword_34': 1,
  'keyword_453': 1,
  'keyword_345': 1,
  'keyword_456': 1,
  'keyword_10': 1,
  'keyword_665': 1,
  'keyword_22': 1,
  'keyword_55': 1,
  'keyword_120': 1,
  'keyword_57': 1,
  'keyword_30': 1,
  'keyword_31': 1,
  'keyword_1000': 0,
  'keyword_234': 0,
  'keyword_77': 0,
  'keyword_976': 0,
  'keyword_987': 0,
  'keyword_222': 0,
  'keyword_543': 0},
 {'name': 'b',
  'type': 'bottom',
  'vendor': 'addidas',
  'color': 'blue',
  'size': 'L',
  'cat_1': 1,
  'cat_2': 1,
  'cat_5': 1,
  'cat_7': 1,
  'cat_10': 0,
  'cat_3': 0,
  'cat_4': 0,
  'keyword_1000': 1,
  'keyword_234': 1,
  'keyword_543': 1,
  'keyword_77': 1,
  'keyword_976': 1,
  'keyword_22': 1,
  'keyword_665': 1,
  'keyword_987': 1,
```

```
  'keyword_222': 1,
  'keyword_31': 1,
  'keyword_34': 0,
  'keyword_453': 0,
  'keyword_456': 0,
  'keyword_10': 0,
  'keyword_55': 0,
  'keyword_120': 0,
  'keyword_345': 0,
  'keyword_30': 0,
  'keyword_57': 0},
 {'name': 'c',
  'type': 'top',
  'vendor': 'nike',
  'color': 'blue',
  'size': 'M',
  'cat_1': 1,
  'cat_2': 1,
  'cat_4': 1,
  'cat_5': 1,
  'cat_10': 1,
  'cat_3': 0,
  'cat_7': 0,
  'keyword_1000': 1,
  'keyword_10': 1,
  'keyword_234': 1,
  'keyword_543': 1,
  'keyword_77': 1,
  'keyword_976': 1,
  'keyword_22': 1,
  'keyword_120': 1,
  'keyword_665': 1,
  'keyword_987': 1,
  'keyword_222': 1,
  'keyword_31': 1,
  'keyword_34': 0,
  'keyword_453': 0,
  'keyword_456': 0,
  'keyword_55': 0,
  'keyword_345': 0,
  'keyword_30': 0,
  'keyword_57': 0}]
```

In [3]:
```
import pandas as pd

item_df = pd.DataFrame.from_dict(items).set_index('name').drop(columns
=['color', 'size'])
item_df = pd.get_dummies(item_df)
item_df
```

Out[3]:

	cat_1	cat_10	cat_2	cat_3	cat_4	cat_5	cat_7	keyword_10	keyword_1000	keyword_
name										
a	1	0	1	1	1	1	0	1	0	
b	1	0	1	0	0	1	1	0	1	
c	1	1	1	0	1	1	0	1	1	

3 rows × 30 columns

In [4]:
```
item_tp = item_df.transpose()
item_tp
```

Out[4]:

name	a	b	c
cat_1	1	1	1
cat_10	0	0	1
cat_2	1	1	1
cat_3	1	0	0
cat_4	1	0	1
cat_5	1	1	1
cat_7	0	1	0
keyword_10	1	0	1
keyword_1000	0	1	1
keyword_120	1	0	1
keyword_22	1	1	1
keyword_222	0	1	1
keyword_234	0	1	1
keyword_30	1	0	0
keyword_31	1	1	1
keyword_34	1	0	0

keyword_345	1	0	0
keyword_453	1	0	0
keyword_456	1	0	0
keyword_543	0	1	1
keyword_55	1	0	0
keyword_57	1	0	0
keyword_665	1	1	1
keyword_77	0	1	1
keyword_976	0	1	1
keyword_987	0	1	1
type_bottom	0	1	0
type_top	1	0	1
vendor_addidas	1	1	0
vendor_nike	0	0	1

In [5]:
```
from sklearn.metrics import jaccard_similarity_score
item_names = item_tp.columns.values
possible_pairs = [(item_names[i],item_names[j]) for i in range(len(ite
m_names)) for j in range(i+1, len(item_names))]
for pair in possible_pairs:
    print('Similarity of {} <-> {} = {}'.format(pair[0].upper(), pair[
1].upper(), jaccard_similarity_score(item_tp[pair[0]].values, item_tp[
pair[1]].values)))
```

```
Similarity of A <-> B = 0.3
Similarity of A <-> C = 0.4
Similarity of B <-> C = 0.7
```

Now consider a dataset of 100K products; what technique(s)/algorithm(s) can you propose to efficiently calculate this similarity between this huge dataset every pair? Let's assume we don't have any limitations on the CPU processors or memory.

- If we just want to find similar products rather than pairs we could have used clustering algorithm, but if we want to find pairs then clustering would not work.
- Comparing manually each pair would make us go through (100,000 Choose 2) = 5B pairs which would be both very expensive
- So when we have 100K products the best strategy to use Locality Sensitive Hashing with Jaccard similarity to find similar pairs.

Write a pseudo-code for your solution(s) and define your preferred data structures.

- Firstly we need to create a sparse matrix from the given dataset
- Using minhashing we can create signature matrix from this datasets
- Choose a threshold t where 0<t<1 where we will choose to pairs are similar if they have jaccard similarity over t
- Use locality sensitivity hashing to find similar pairs
  - Initially choose as many possible hash function to create signature matrix
  - Each row would be one signature matrix.
  - So if we have N has we would have a matrix N x 100,000
  - Now divide N rows into b bands where each band contains r rows. N = b * r
  - For each band we hash all the columns to K buckets
  - For any band if two different columns hash into one bucket we consider items whose parts are these columns are as similar pair.
  - We can tune our numbers of bands and rows to reduce false positive and true negetive
- An implementation of LSH was done by me back in 2017 as an SFU academic projects details of that can be accessed here https://github.com/mdrmuhaimin/Implement_LSH (https://github.com/mdrmuhaimin/Implement_LSH)
- My understand of LSH is based on Prof. Jeff Ulman's lecture which can be found here
  - Part 1 (https://youtu.be/bQAYY8INBxg)
  - Part 2 (https://youtu.be/MaqNINSY4gc)

```
In [ ]:
```