Assignment4

Made by Yu Zhang a1795409

Ex1:

The Code is in appendix

```
print(distance.cosine(B_bool,C_
```

```
[1. 1. 0. 1. 1. 0. 1. 1.]
[0. 1. 1. 1. 1. 1. 1. 0.]
[1. 0. 1. 1. 0. 1. 1. 1.]
```

1. The Boolean matrix is
   Jaccard distance between AB: 0.5, AC: 0.5, BC: 0.5.
2. Cosine distance of AB: 0.333, AC: 0.333, BC: 0.333

```
            num = 1;
        result = np.append(result,num);
    return result;

A_bool = to_bool(A);
B_bool = to_bool(B);
C_bool = to_bool(C);

D_AB = jaccard(A_bool,B_bool)
D_AC = jaccard(A_bool,B_bool)
D_BC = jaccard(B_bool,C_bool)
print(A_bool)
print(B_bool)
print(C_bool)
print(D_AB)
print(D_AC)
print(D_BC)

print(distance.cosine(A_bool,B_bool))
print(distance.cosine(A_bool,C_bool))
print(distance.cosine(B_bool,C_bool))
```

```
[1. 1. 0. 1. 1. 0. 1. 1.]
[0. 1. 1. 1. 1. 1. 1. 0.]
[1. 0. 1. 1. 0. 1. 1. 1.]
0.5
0.5
0.5
0.33333333333333337
0.33333333333333337
0.33333333333333337
```

```
[1. 1. 0. 1. 0. 0. 1. 0.]
[0. 1. 1. 1. 0. 0. 0. 0.]
[0. 0. 0. 1. 0. 1. 1. 1.]
```

3. New Boolean matrix is
   Jaccard distance between AB: 0.6 AC: 0.667 BC: 0.833
4. Cosine distance of AB: 0.423, AC: 0.5, BC: 0.711

```
        if (num >= 3):
            num = 1;
        else:
            num = 0;
        result = np.append(result,num);
    return result;

A_clu = to_cluster(A);
B_clu = to_cluster(B);
C_clu = to_cluster(C);
print(A_clu)
print(B_clu)
print(C_clu)

print(jaccard(A_clu,B_clu))
print(jaccard(A_clu,C_clu))
print(jaccard(B_clu,C_clu))

print(distance.cosine(A_clu,B_clu))
print(distance.cosine(A_clu,C_clu))
print(distance.cosine(B_clu,C_clu))
```

```
[1. 1. 0. 1. 0. 0. 1. 0.]
[0. 1. 1. 1. 0. 0. 0. 0.]
[0. 0. 0. 1. 0. 1. 1. 1.]
0.6
0.6666666666666666
0.8333333333333334
0.42264973081037416
0.5
0.7113248654051871
```

5. Normalized matrix:

```
[ 0.66666667  1.66666667  0.          1.66666667 -2.33333333  0.
 -0.33333333 -1.33333333]
[ 0.          0.66666667  1.66666667  0.66666667 -1.33333333 -0.3
3333333
 -1.33333333  0.        ]
[-1.  0. -2.  0.  0.  1.  2.  0.]
```

6. Cosine distance between AB: 0.416, AC: 1.115, BC: 1.740

In [3]:
```
def to_norm(array):
    result = np.array([])
    mean = array[np.nonzero(array)].mean()
    for num in array:
        if (num != 0):
            num = num - mean;
        result = np.append(result,num);
    return result;
A_norm = to_norm(A)
B_norm = to_norm(B)
C_norm = to_norm(C)
print(A_norm)
print(B_norm)
print(C_norm)

print(distance.cosine(A_norm,B_norm))
print(distance.cosine(A_norm,C_norm))
print(distance.cosine(B_norm,C_norm))
```

```
[ 0.66666667  1.66666667  0.          1.66666667 -2.33333333  0.
 -0.33333333 -1.33333333]
[ 0.          0.66666667  1.66666667  0.66666667 -1.33333333 -0.3
3333333
 -1.33333333  0.        ]
[-1.  0. -2.  0.  0.  1.  2.  0.]
0.41569345253185686
1.1154700538379252
1.739573996953447
```

Ex2:
The Code is in appendix
3. Randomized algorithm Under 0.5 random possibility and 0.5 support

Mushroom.dat:

```
a1795409@LAPTOP-PP7DT1HP:~/Test/MBDa4$ python3 randomize.py -f mushroom.dat
8125  lines in total.
sample size =  4073
The frequent set with size 1 is:
the count is: 13
The frequent set with size 2:
count: 43
The frequent set with size 3:
count: 59
The frequent set with size 4:
count: 36
The frequent set with size 5:
count: 8
The frequent set with size 6:
count: 0
Total running time: 1.8615732192993164
```

Chess.dat

```
a1795409@LAPTOP-PP7DT1HP:~/Test/MBDa4$ python3 randomize.py -f chess.dat
3197  lines in total.
sample size =  1570
The frequent set with size 1 is:
the count is: 37
The frequent set with size 2:
count: 526
The frequent set with size 3:
count: 3974
The frequent set with size 4:
count: 18735
```

Connect.dat

```
a1795409@LAPTOP-PP7DT1HP:~/Test/MBDa4$ python3 randomize.py -f connect.dat
67558  lines in total.
sample size =  33740
The frequent set with size 1 is:
the count is: 38
The frequent set with size 2:
count: 630
```

Pumsb.dat:

```
a1795409@LAPTOP-PP7DT1HP:~/Test/MBDa4$ python3 randomize.py -f pumsb.dat
49047  lines in total.
sample size =  24580
The frequent set with size 1 is:
the count is: 52
The frequent set with size 2:
count: 982
```

Pumsb_star.dat:

```
a1795409@LAPTOP-PP7DT1HP:~/Test/MBDa4$ python3 randomize.py -f pumsb_star.dat
49047  lines in total.
sample size =  24513
The frequent set with size 1 is:
the count is: 27
The frequent set with size 2:
count: 94
The frequent set with size 3:
count: 172
The frequent set with size 4:
count: 191
```

T10I4D100k:

```
a1795409@LAPTOP-PP7DT1HP:~/Test/MBDa4$ python3 randomize.py -f T10I4D100K.dat
100001  lines in total.
sample size =  49980
The frequent set with size 1 is:
the count is: 0
Total running time: 5.072871446609497
```

T40I10D100k:

```
a1795409@LAPTOP-PP7DT1HP:~/Test/MBDa4$ python3 randomize.py -f T40I10D100K.dat
100001  lines in total.
sample size =  50164
The frequent set with size 1 is:
the count is: 0
Total running time: 6.748476266860962
```

4. randomized algorithm with different sample sizes:

1%:

```
a1795409@LAPTOP-PP7DT1HP:~/Test/MBDa4$ python3 randomize.py -p 0.01
8125  lines in total.
sample size =  91
The frequent set with size 1 is:
the count is: 14
The frequent set with size 2:
count: 53
The frequent set with size 3:
count: 86
The frequent set with size 4:
count: 66
The frequent set with size 5:
count: 23
The frequent set with size 6:
count: 3
The frequent set with size 7:
count: 0
Total running time: 0.10032129287719727
```

2%:

```
a1795409@LAPTOP-PP7DT1HP:~/Test/MBDa4$ python3 randomize.py -p 0.02
8125  lines in total.
sample size =  183
The frequent set with size 1 is:
the count is: 15
The frequent set with size 2:
count: 49
The frequent set with size 3:
count: 69
The frequent set with size 4:
count: 48
The frequent set with size 5:
count: 16
The frequent set with size 6:
count: 2
The frequent set with size 7:
count: 0
Total running time: 0.18019413948059082
```

5%:

```
a1795409@LAPTOP-PP7DT1HP:~/Test/MBDa4$ python3 randomize.py -p 0.05
8125  lines in total.
sample size =  413
The frequent set with size 1 is:
the count is: 14
The frequent set with size 2:
count: 48
The frequent set with size 3:
count: 72
The frequent set with size 4:
count: 51
The frequent set with size 5:
count: 15
The frequent set with size 6:
count: 1
The frequent set with size 7:
count: 0
Total running time: 0.3159055709838867
```

10%:

```
a1795409@LAPTOP-PP7DT1HP:~/Test/MBDa4$ python3 randomize.py -p 0.1
8125  lines in total.
sample size =  784
The frequent set with size 1 is:
the count is: 14
The frequent set with size 2:
count: 43
The frequent set with size 3:
count: 56
The frequent set with size 4:
count: 34
The frequent set with size 5:
count: 8
The frequent set with size 6:
count: 0
Total running time: 0.3863530158996582
```

It is clear that with the sample size increasing, the running time and frequency of each size set are increasing too.

EX3:
1.

Greedy Algorithm:

Greedy algorithm intends to maximize some particular functions of the input and past. In the online advertisement example, for maximizing the income of the search engine provider, the greedy algorithm will always give the finite advertisement slots to the advertiser who offer the highest bid with related query, until the advertiser runs out of budgets by engine users' clicking visit.

Balanced Algorithm:
Compared with the greedy algorithm, the balanced algorithm will give the slots to the advertiser who has higher remain budgets which related to the assigned query. It usually has higher competitive ratio to the greedy algorithm.

Competitive ratio:
The competitive ratio is an online matching algorithm in the worst-case performance compare with the off-line ideally matching algorithm. Since the input query online is similar like the stream data and unpredictable. The higher competitive ratio means the better performance of online algorithms. The competitive ratio can only tend to 1.

2. The result 4 of 6 queries with greedy algorithm:
   In Greedy algorithm if A offered highest bid, then C and B.
   The first 2x should give to A, then 2y give to C, at last 2c but C out of budget and B didn't bid for the query so no one can be assigned. The competitive ratio is 4/6.

   The sequence of queries which few as half:

In Greedy algorithm if C offered highest price.
The sequence XXZZ coming, the first 2X will assign to C, and C out of budgets. Then no one has remained budgets for coming 2Z. the competitive ratio is 1/2.