

1조 Q&A 🤔

이정인, 신민경 , 전성원 , 김덕영

Q. 수업시간에 다뤘던 알고리즘은 오버랩 카운트를 재사용하지 않았는데,
PPJoin이 맞는지 궁금합니다.

```
for i in range(len(R)): # 후보 검증 후 S에 추가
    (y_id, y) = R[i]
    if A[i] == 0: continue # prefix 안에 공통 토큰이 하나도 없다면 제외
    a = math.ceil((len(x)+len(y))*r/(r+1))
```

검증 코드 부분에서

count = 0 이면, 공통 토큰을 더이상 count하지 않는 방법으로 재사용합니다.

저희는 최종적으로 가장 효율적인 알고리즘을 APJoin로 선정해서 구현한 것이므로 PPJoin 방식과는 다릅니다.

Q. Candidate Pair을 줄이는 과정에서, 실제 threshold를 넘을 후보 pair의 손실은 없었나요?

[수도 코드]

```
11 for i = 1 to max_probe_prefix do
12   w ← x[i];           // 레코드 x의 토큰 (x, i)
13   for each (y, j) ∈ I_w do      // 레코드 y의 토큰 (y, j)
14     if |y| < t · |x|           // 현재 노드 이후 삭제
15       (y, j)와 연결된 노드들 삭제;
16     else if j > prefix_y[|x| - |y| + 1]
17       (y, j) 삭제;           // 현재 노드 삭제
18     else if i > prefix_x[|x| - |y| + 1]
19       continue;           // 다음 노드로 이동
20     else
21       A[y] = A[y] + 1;      // candidate generation
```

$$O(x, y) \geq \alpha = \frac{t}{1+t} (|x| + |y|)$$

$$\alpha \leq |y|, \quad \frac{t}{1+t} (|x| + |y|) \leq |y|$$
$$t|x| \leq |y|$$

Length filtering과 prefix filtering 에서는

후보가 될 가능성이 전혀 없는 경우만 제외하기 때문에 후보 Pair에 대한 손실이 없습니다.

Q. 혹시 직접 생각한 알고리즘이 있나요?

```
for i = 1 to max_index_prefix do
    w ← x[i];
    Iw = Iw ∪ {(x, i)};    // 토큰 (x, i)를 inverted index에
    VerifyZip(x, A, α);      // verification phase
return S
```

```
for i in range(len(R)): # 후보 검증 후 S에 추가
    (y_id, y) = R[i]

    if A[i] == 0: continue # prefix 안에 공통 토큰이 하나도 없다면 제외

    a = math.ceil((len(x)+len(y))*r/(r+1))

    (p,q,cnt) = (0,0,0)

    while(p<len(x) and q<len(y)): # 공통 토큰 개수 카운트
        if x[p] == y[q]:
            cnt = cnt+1
            p = p+1
            q = q+1
        elif cnt+min(len(x)-p-1, len(y)-q-1) < a: break # positional filtering
        elif x[p] < y[q]: p = p+1
        else: q = q+1
    if cnt >= a and y_id not in x: # x와 y가 이미 친구인 경우는 제외
        S.append((x_id, y_id))
```

논문에서 제시된 APJoin 알고리즘은 검증에 관한 VerifyZip 코드에 대한 설명과 구현이 안되어 있었는데,

- 1) 이에 대해 저희는 VeritifyZip 코드를 직접 구현하였습니다.
- 2) 공통토큰을 세기 위해 $O(n^2)$ 방법 대신 $O(n)$ 방법으로 개선하는 방향으로 코드를 생각하여 구현했습니다.
- 3) 이미 친구인 경우에 어떻게 후보군을 제외시킬 지에 대해 직접 고민해봤습니다.

Q. 최종 결과가 어떻게 나왔는지 확인 가능할까요?

1	1	2
2	2	3
3	2	5
4	4	3
5	6	3
6	4	6

```
# when r = 0.1
[(5, 1), (2, 6), (2, 4), (3, 1), (3, 5)]
duration : 0.0010955333709716797
# when r = 0.6
[(5, 1)]
duration : 0.0004329681396484375
# when r = 0.7
[(5, 1)]
duration : 0.0004551410675048828
# when r = 0.8
[(5, 1)]
duration : 0.0003483295440673828
# when r = 0.9
[(5, 1)]
duration : 0.0003559589385986328
```

Simple Test Case 를 생성하여 결과를 돌려봤습니다.

Q. 자료조사 열심히 하신 것 잘 봤습니다. 그 많은 알고리즘들을 다 적용해 보신건가요?

논문 내 4가지 Join들의 성능비교

	Comp	1) Cand	Join	Time(sec)
PPJoin 2)	4,132,651	1,356,337	3,683	2.106
PPJoin+	4,132,651	11,905	3,683	1.810
MPJoin	2,200,083	1,356,628	3,683	0.978
APJoin	1,378,845	1,357,015	3,683	0.668

1) PPJoin vs. PPJoin+: 비교연산 수는 동일, 추출된 조인후보 쌍은 큰 차이

2) PPJoin vs. (MPJoin & APJoin): 비교연산 수는 큰 차이, 추출된 조인후보 쌍은 차이 없음

BruteForce 와 수업시간에 간단한 Similarity Join 관련 알고리즘에 대해 구현을 해보며 이해를 했었고, PPJoin과 PPJoin+, MPJoin 알고리즘에 대한 수도코드 분석을 통해 논문을 이해하고 정리했습니다.

논문에서도 성능검증이 완료된 표를 통해 알 수 있었듯이 최종적으로 가장 효율적인 알고리즘이 'APJoin' 을 최종 구현했습니다.

Q. 문제에서 해결해야 하는 친구 추천 코드 구현 시, pyspark는 어떤 식으로 이용했나요?

```
In [1]: import findspark
        findspark.init()
```

```
In [2]: from pyspark import SparkContext
        from pyspark.sql import SQLContext, SparkSession

        sc = SparkContext("local", "pbl-1")
        spark = SparkSession(sc)
```

```
In [3]: data = sc.textFile("facebook_combined.txt")

        parse_data = data.flatMap(lambda line: line.split('\n')) \
                           .map(lambda l: l.split(' ')).map(lambda l: [(int(l[0]),int(l[1])),(int(l[1]),int(l[0]))]).flatMap(lambda t:
```

```
In [4]: group = parse_data.groupByKey().mapValues(list)
        new_group = group.flatMap(lambda l: [(l[0],(len(l[1]), sorted(l[1])))])
        #print(new_group.collect())
```

```
In [5]: order = new_group.sortBy(lambda v: v[1])
        R = order.mapValues(lambda v: v[1]).collect()
        #print(R)
```


Q. 문제에서 해결해야 하는 친구 추천 코드 구현 시, pyspark는 어떤 식으로 이용했나요?

4: Codes & Results

동적 Prefix Filtering/Inverted list 작성

```
Algorithm: APJoin(R, t)
1 S ← ∅;
2 I_w ← ∅ (1 ≤ w ≤ |U|); // initialize inverted index
3 for each x ∈ R do
4   A ← empty map from record id to int;
5   max_probe_prefix ← |x| - ⌈t · |x|⌉ + 1; // Eq. (5)
6   max_index_prefix ← |x| - ⌈2|x| · t/(t+1)⌉ + 1; // Eq. (6)
7   for i = 1 to max_probe_prefix do
8     α = ⌈(|x|+|x|-i+1) · t/(t+1)⌉; // |y|=|x|-i+1
9     prefix_x[i] = |x|-α+1; // Eq. (3)
10    prefix_y[i] = (|x|-i+1)-α+1; // Eq. (4)
11   for i = 1 to max_probe_prefix do
12     w ← x[i]; // 레코드 x의 토큰 (x, i)
13     for each (y, j) ∈ I_w do // 레코드 y의 토큰 (y, j)
14       if |y| < t · |x| // 현재 토큰 이후 삭제
15         (y, j)와 연결된 노드들 삭제;
16       else if j > prefix_y[|x|-|y|+1]
17         (y, j) 삭제; // 현재 토큰 삭제
18       else if i > prefix_x[|x|-|y|+1]
19         continue; // 다음 토큰으로 이동
20       else
21         A[y] = A[y] + 1; // candidate generation
22   for i = 1 to max_index_prefix do
23     w ← x[i];
24     I_w = I_w ∪ {(x, i)}; // 토큰 (x, i)를 inverted index에
25   VerifyZip(x, A, α); // verification phase
26 return S
```

```
for i in range(1,max_probe_prefix+1):
    w = x[i-1]
    w_idx = set_idx[w]

    for ((y_id, y),j) in I[w_idx]:
        if len(y) < r*len(x):
            I[w_idx].remove(((y_id,y),j))
        elif j > prefix_y[len(x)-len(y)+1]:
            I[w_idx].remove(((y_id,y),j))
        elif i > prefix_x[len(x)-len(y)+1]: continue
        else:
            A[set_idx[y_id]] = A[set_idx[y_id]] + 1 # 후보 확정
```

```
for i in range(1,max_index_prefix+1): # x의 원소에 대한 inverted list 추가
    w = x[i-1]
    w_idx = set_idx[w]
    I[w_idx].append(((x_id,x),i))
```

조인 후보 쌍 검증

```
Algorithm: APJoin(R, t)
1 S ← ∅;
2 I_w ← ∅ (1 ≤ w ≤ |U|); // initialize inverted index
3 for each x ∈ R do
4   A ← empty map from record id to int;
5   max_probe_prefix ← |x| - ⌈t · |x|⌉ + 1; // Eq. (5)
6   max_index_prefix ← |x| - ⌈2|x| · t/(t+1)⌉ + 1; // Eq. (6)
7   for i = 1 to max_probe_prefix do
8     α = ⌈(|x|+|x|-i+1) · t/(t+1)⌉; // |y|=|x|-i+1
9     prefix_x[i] = |x|-α+1; // Eq. (3)
10    prefix_y[i] = (|x|-i+1)-α+1; // Eq. (4)
11   for i = 1 to max_probe_prefix do
12     w ← x[i]; // 레코드 x의 토큰 (x, i)
13     for each (y, j) ∈ I_w do // 레코드 y의 토큰 (y, j)
14       if |y| < t · |x| // 현재 토큰 이후 삭제
15         (y, j)와 연결된 노드들 삭제;
16       else if j > prefix_y[|x|-|y|+1]
17         (y, j) 삭제; // 현재 토큰 삭제
18       else if i > prefix_x[|x|-|y|+1]
19         continue; // 다음 토큰으로 이동
20       else
21         A[y] = A[y] + 1; // candidate generation
22   for i = 1 to max_index_prefix do
23     w ← x[i];
24     I_w = I_w ∪ {(x, i)}; // 토큰 (x, i)를 inverted index에
25   VerifyZip(x, A, α); // verification phase
26 return S
```

```
for i in range(len(R)): # 후보 검증 후 S에 추가
    (y_id, y) = R[i]

    if A[i] == 0: continue # prefix 안에 공통 토큰이 하나도 없다면 제외

    a = math.ceil((len(x)+len(y))*r/(r+1))

    (p,q,cnt) = (0,0,0)
```

```
while(p<len(x) and q<len(y)): # 공통 토큰 개수 카운트
    if x[p] == y[q]:
        cnt = cnt+1
        p = p+1
        q = q+1
    elif cnt+min(len(x)-p-1,len(y)-q-1) < a: break # positional filtering
    elif x[p] < y[q]: p = p+1
    else: q = q+1
if cnt >= a and y_id not in x: # x와 y가 이미 친구인 경우는 제외
    S.append((x_id, y_id))
```

4: Codes & Results

변수 선언/α 값·prefix 범위 계산

```
Algorithm: APJoin(R, t)
1 S ← ∅;
2 I_w ← ∅ (1 ≤ w ≤ |U|); // initialize inverted index
3 for each x ∈ R do
4   A ← empty map from record id to int;
5   max_probe_prefix ← |x| - ⌈t · |x|⌉ + 1; // Eq. (5)
6   max_index_prefix ← |x| - ⌈2|x| · t/(t+1)⌉ + 1; // Eq. (6)
7   for i = 1 to max_probe_prefix do
8     α = ⌈(|x|+|x|-i+1) · t/(t+1)⌉; // |y|=|x|-i+1
9     prefix_x[i] = |x|-α+1; // Eq. (3)
10    prefix_y[i] = (|x|-i+1)-α+1; // Eq. (4)
11   for i = 1 to max_probe_prefix do
12     w ← x[i]; // 레코드 x의 토큰 (x, i)
13     for each (y, j) ∈ I_w do // 레코드 y의 토큰 (y, j)
14       if |y| < t · |x| // 현재 토큰 이후 삭제
15         (y, j)와 연결된 노드들 삭제;
16       else if j > prefix_y[|x|-|y|+1]
17         (y, j) 삭제; // 현재 토큰 삭제
18       else if i > prefix_x[|x|-|y|+1]
19         continue; // 다음 토큰으로 이동
20       else
21         A[y] = A[y] + 1; // candidate generation
22   for i = 1 to max_index_prefix do
23     w ← x[i];
24     I_w = I_w ∪ {(x, i)}; // 토큰 (x, i)를 inverted index에
```

```
r_set = [0.6, 0.7, 0.8, 0.9]
set_idx = {id:i for i,(id,x) in enumerate(R)}

times_ = []
length_ = []

for r in r_set:
    print("# when r = ",r)
    start = time.time()

    S = []
    I = [[] for _ in range(len(R))] # inverted list
```

```
for (x_id, x) in R:
    max_probe_prefix = len(x) - math.ceil(r*len(x)) + 1
    max_index_prefix = len(x) - math.ceil(2*len(x)*r/(r+1)) + 1
    prefix_x = [0]
    prefix_y = [0]
    A = [0 for _ in range(len(R))] # overlap score for y

    for i in range(1,max_probe_prefix+1):
        a = math.ceil((2*len(x)-i+1)*r/(r+1))
        prefix_x.append(len(x)-a+1)
        prefix_y.append(len(x)-i+1-a+1)
```


Q. 문제에서 해결해야 하는 친구 추천 코드 구현 시, pyspark는 어떤 식으로 이용했나요?

RDD로 구현한 코드

```
for (x_id, x) in R: # 순서대로 inverted list를 작성해야 하므로 동기
    A = [0 for _ in range(len(R))]
```

```
max_probe_prefix = len(x) - math.ceil(r*len(x)) + 1
max_index_prefix = len(x) - math.ceil(2*len(x)*r/(r+1)) + 1
```

```
prefix_x = sc.parallelize([i for i in range(max_probe_prefix+1)]) \
    .map(lambda t: len(x)-(math.ceil((2*len(x)-t+1)*r/(r+1)))+1).collect()
prefix_y = sc.parallelize([i for i in range(max_probe_prefix+1)]) \
    .map(lambda t: len(x)-t+1-(math.ceil((2*len(x)-t+1)*r/(r+1)))+1).collect()
```

```
for i in range(1,max_probe_prefix+1):
    w = x[i-1]
    w_idx = set_idx[w]
```

```
inverted_list = sc.parallelize(l[w_idx]).filter(lambda t: len(t[0][1]) >= r*len(x) and t[1] <= prefix_y[len(x)-len(t[0][1])+1])
l[w_idx] = inverted_list.collect()
```

```
candidate = inverted_list.filter(lambda t: i <= prefix_x[len(x)-len(t[0][1])+1]).collect()
for item in candidate:
    item_id = item[0][0]
    A[set_idx[item_id]] = A[set_idx[item_id]]+1
```

duration : 699.9134256839752
when r = 0.7

Q. Positional filtering에서 후보군 제거 과정에서, 어떻게 제거한 건지 궁금합니다.

3: Algorithm Research - Positional Filtering

: Overlap의 최대크기 $\geq \alpha$ 를 만족하지 못하는 레코드를 조인후보 쌍 생성 전에 제외한다.

레코드 x, y 의 공통토큰이 각각 i 번째, j 번째 있다고 하자.

Overlap의 최대크기 = 현재 Overlap 크기 + $\min(\text{len}(x)-i, \text{len}(y)-j)$

prefix filtering은 prefix 범위 내에서는

$O(x,y) \geq \alpha$ 를 만족하지 못하더라도 조인후보 쌍으로 선택한다.

-> 불필요한 조인후보 쌍이 많이 생길 수 있다.

\therefore positional filtering 이용하면, 생성되는 조인후보의 쌍을 많이 줄일 수 있다.