

VF2 : A (Sub)Graph Isomorphism Algorithm for Matching Large Graphs

1. 환경 및 실행방법

- python >= 3.10

VF2 Matching

- `./run.sh <pattern graph> <target graph> <output file>`
- 디버깅 모드
 - Candidate Pairs, Feasibility Failure, Rollback 등 동작에 대한 자세한 로그를 볼 수 있습니다.
 - `./run.sh tests/input_g1.txt tests/input_g2.txt output1.txt --debug`

Checker

- `./run_checker.sh <pattern graph> <target graph> <output file>`
-

2. Input/Output Format

Input Format

Input graph 는 Directed Graph 이며, 아래와 같이 nodes와 edges로 나누어져 있는 text 파일 형태입니다.

#nodes : 각 노드의 고유 ID (int)와 라벨(label) 을 정의합니다.

- `<Node ID> <Node Label>`

#edges : 노드 간의 매핑관계 (edge)을 정의합니다.

- `<출발 Node ID> <도착 Node ID>`

```
#nodes
1 A
2 B
```

```
3 C
```

```
#edges
```

```
1 2
```

```
2 3
```

Output Format

- 첫번째 line : Isomorphic 여부를 나타내는 값 (True, False)
- 이 후 : Matched Node ID Pairs

```
True
```

```
1 2
```

```
2 1
```

```
3 3
```

3. Testcase 및 동작 설명

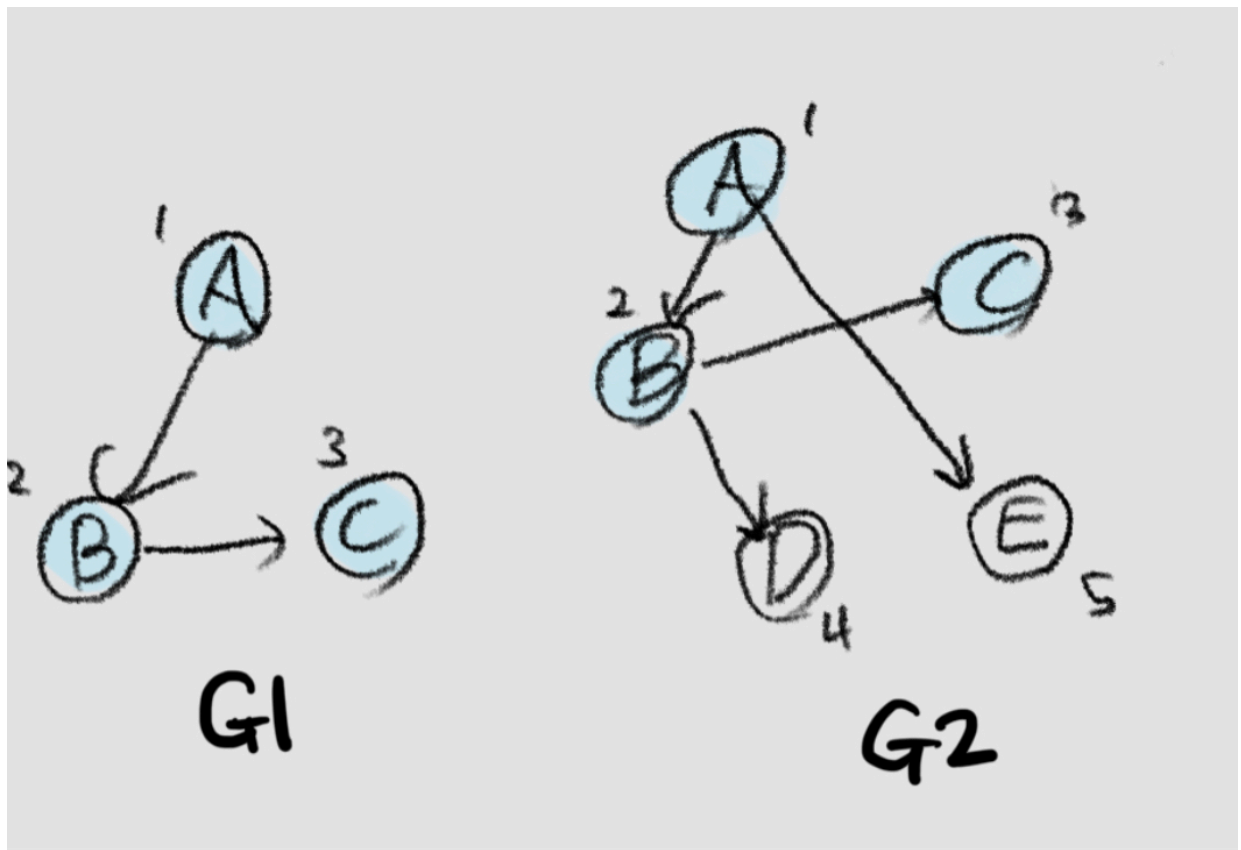
Subgraph isomorphism 과 관련된 Matched, UnMatched, Disconnected, Cycle Graph 등 다양한 테스트를 진행하였습니다.

각 TC 에 대한 동작 설명은 다음과 같습니다.

Testcase 1 : Matched Case

: G1이 G2의 Sub-Graph 인 케이스입니다. 해당 케이스에서는 Candidate Pair 생성과 관련하여 동작을 확인할 수 있습니다.

- Candidate Pairs 생성 로직
 - 초기 그래프의 경우, 모든 노드에 대해 Candidate를 생성
 - 이후, Matched 를 진행하며 현재 그래프에서 out되는 노드인 T_{out} 으로부터 Candidate 를 생성하고, T_{out} 이 존재하지 않을 경우 T_{in} 으로부터 Candidate를 생성합니다.
 - 이 TC에서는 (3,3) from all node pairs -> (2,2) from T_{in} -> (1,1) from T_{in} 순서로 Candidate 가 생성되도록 구성하였습니다.



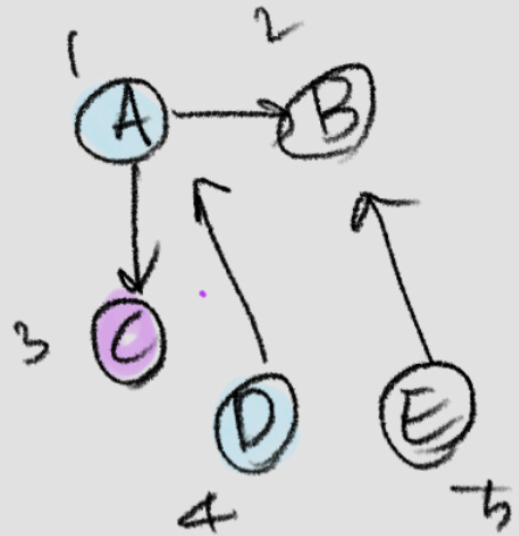
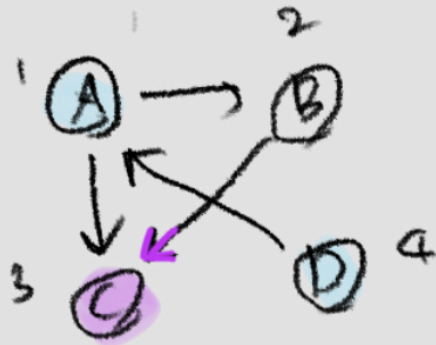
```
Candidate pair: 3, 3
Candidate pair: 2, 2
Candidate pair: 1, 1
[(Node(3, 'C'), Node(3, 'C')), (Node(2, 'B'), Node(2, 'B')), (Node(1, 'A'), Node(1, 'A'))]
```

Testcase 2 : UnMatched Case

: G1이 G2의 Sub-Graph가 아닌 케이스입니다. 해당 테스트 케이스를 통해 다양한 Feasibility 위반 동작을 살펴볼 수 있습니다.

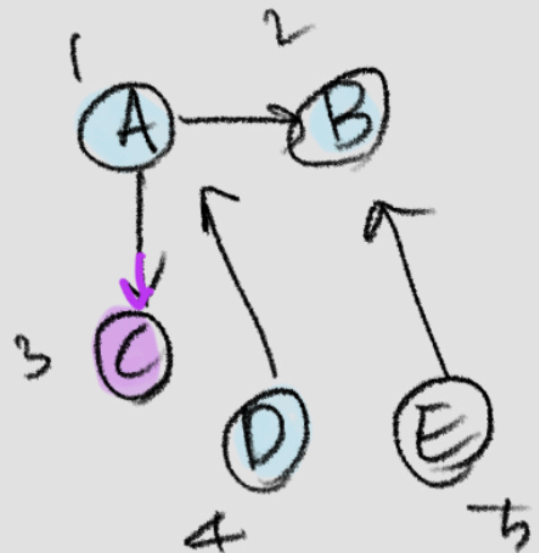
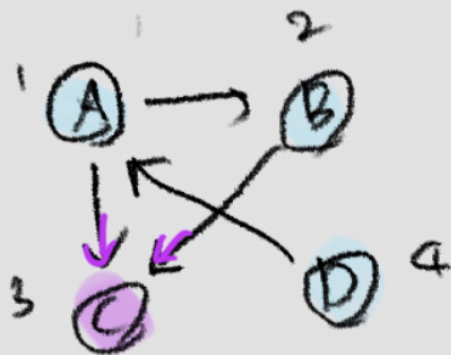
- Match loop 상세
 - step1) (4,4) : 구조적 feasibility와 semantic feasibility ($D=D$) 모두 만족 -> add matched M
 - step2) (1,1) : add matched M
 - step3) (3,3) : R_{in} 위반
 - Node 3으로 들어오는 in-node의 개수 ($T_{in} \cap Pred$) 가 $G1 > G2$ ($1 > 0$)

R_{in} 위배 (3,3)



- step4) (3,2) R_{in} 위배
- step5) (2,3) 구조적 feasibility는 만족하나, label 이 다름 ($B \neq C$)
- step6) (2,2) add matched M
- step7) (3,3) R_{pred} 위배
 - Matched 된 Predecessor 조건을 만족하지 않음 $A, B \notin A$

R_{pred} 위배 (3,3)



- 이후 step 생략
- 최종 Unmatched (non-isomorphic case)

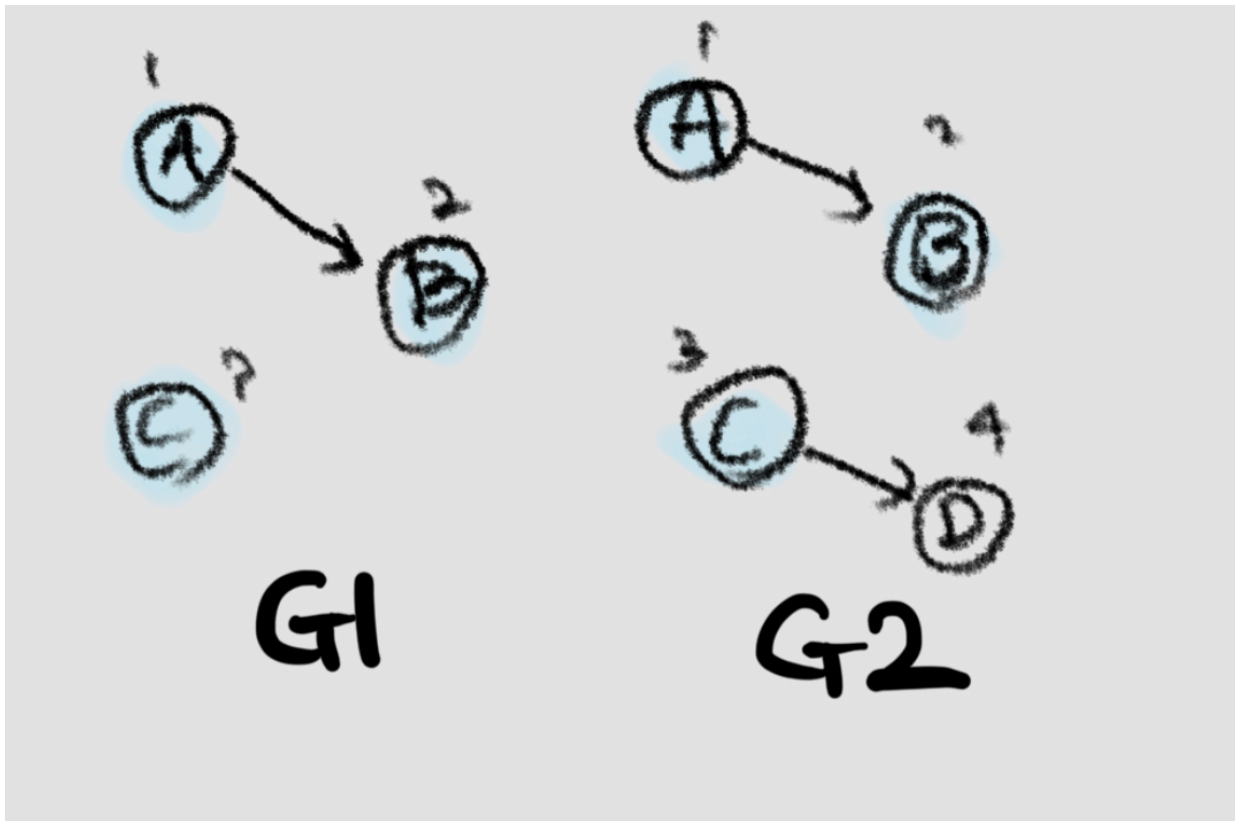
```

da02 ~./workspace/vf2 [ main ± python3 main.py tests/input2_g1.txt tests/input2_g2.txt output.txt --debug
Candidate pair: 4, 4
Candidate pair: 1, 1
Candidate pair: 2, 3
  Label (Semantic Attribute) check failed
Candidate pair: 2, 2
Candidate pair: 3, 3
  R_pred check failed
Candidate pair: 3, 3
  R_new check failed 1 > 0
Candidate pair: 3, 2
  R_new check failed 1 > 0
Candidate pair: 4, 1
  Label (Semantic Attribute) check failed
Candidate pair: 4, 3
  R_new check failed 2 > 1
Candidate pair: 4, 2
  R_new check failed 2 > 1
Candidate pair: 4, 5
  Label (Semantic Attribute) check failed
Candidate pair: 1, 4
  Label (Semantic Attribute) check failed
Candidate pair: 1, 1
Candidate pair: 4, 4
Candidate pair: 2, 3
  Label (Semantic Attribute) check failed
Candidate pair: 2, 2
Candidate pair: 3, 3
  R_pred check failed
Candidate pair: 3, 3
  R_new check failed 1 > 0
Candidate pair: 3, 2
  R_new check failed 1 > 0
Candidate pair: 1, 3
  R_new check failed 2 > 1
Candidate pair: 1, 2
  R_new check failed 2 > 1
Candidate pair: 1, 5
  Label (Semantic Attribute) check failed
Candidate pair: 2, 4
  Label (Semantic Attribute) check failed
Candidate pair: 2, 1
  Label (Semantic Attribute) check failed
Candidate pair: 2, 3
  Label (Semantic Attribute) check failed
Candidate pair: 2, 2
Candidate pair: 1, 1
Candidate pair: 4, 4
Candidate pair: 3, 3
  R_pred check failed
Candidate pair: 4, 5
  R_succ check failed
Candidate pair: 1, 5
  Label (Semantic Attribute) check failed
Candidate pair: 2, 5
  Label (Semantic Attribute) check failed
Candidate pair: 3, 4
  Label (Semantic Attribute) check failed
Candidate pair: 3, 1
  Label (Semantic Attribute) check failed
Candidate pair: 3, 3
  R_new check failed 2 > 1
Candidate pair: 3, 2
  R_new check failed 2 > 1
Candidate pair: 3, 5
  Label (Semantic Attribute) check failed
[]

```

Testcase 3 : Matched Case (Disconnected)

- Disconnected Graph 에서 해당 알고리즘이 잘 동작함을 보여주는 Testcase입니다.
- R_{new} 의 경우, Disconnected Graph 의 경우에도 구조적으로 동일한지를 확인하는 feasibility 입니다.



```
da02 ~ /workspace/vf2 [main] python3 main.py tests/input3_g1.txt tests/input3_g2.txt output.txt --debug
Candidate pair: 1, 1
Candidate pair: 2, 2
Candidate pair: 3, 3
[(Node(1, 'A'), Node(1, 'A')), (Node(2, 'B'), Node(2, 'B')), (Node(3, 'C'), Node(3, 'C'))]
```

4. Checker 설명

- `check_label()`, `check_children()` 함수는 `match`에서 짝지어진 `node`의 `label`이 일치하는지, `g1`(smaller graph) `node`의 `children node`가 모두 `g2`(bigger graph) `node`의 `children set`에 포함되는지 확인합니다.
- `check_true()`에서는 이 두 함수를 이용해 `vf2`의 결과가 `true`일 때 `match`가 정확한지 확인합니다.
- `check_false()`에서는 가능한 모든 `node pair`를 만든다. 이를 `check_true`의 `match`에 `input`으로 넣어 `g1`이 `g2`의 `subgraph`가 아닌 것이 맞는지 확인합니다.