**Matching order**

* 우선 특수한 경우로 candidate set size가 1인 vertex의 경우, 해당 vertex에 대해서는 미리 선점을 해 두었다.
* 그 외의 일반적인 경우 DAG ordering을 사용했다. 다만 DAG는 기존 논문과 다른 방법으로 구성했다. Query graph의 vertex를 candidate set size 순으로 정렬하고(같은 경우 degree의 역순), 각 edge의 방향을 정렬된 순서에서 앞에 있는 vertex가 뒤에 있는 vertex로 향하도록 구성했다. 이렇게 구성할 경우 candidate set size가 작은 vertex가 먼저 처리되고 candidate set size가 큰 vertex가 나중에 처리될 확률이 높아진다. DAG ordering에 따라 backtracking을 진행하되 현재 partial embedding에서 extendable vertex가 여러 개인 경우 아래의 우선순위를 따랐다.
* 기본적으로는 현재의 partial embedding에서 extendable candidate set size가 작은 순서대로 매칭이 되도록 하였다. 다만 특수한 상황을 고려하여 아래의 조건이 추가되었다.
* 해당 vertex의 out-degree가 0인 경우(자식이 없는 경우) 이 vertex는 extendable candidate set size가 0이 아닌 이상 현재 extendable candidate set에 들어있는 모든 vertex가 매칭이 가능하고, 이 vertex의 매칭 결과가 다른 vertex에게 영향을 끼치지 않는다. 따라서 이 경우에는 나중에 선택되도록 하였다.
* 해당 vertex의 in-degree가 0인 경우(부모가 없는 경우) 이 vertex의 extendable candidate set은 다른 곳에서 매칭이 진행되어도 변화하지 않는다. 따라서 나중에 선택된다고 하여 상황이 더 나아지거나 하지 않으므로 해당 vertex가 우선적으로 선택되어 다른 vertex들의 extendable candidate set에 미리 영향을 끼치도록 하였다.
* 모든 vertex를 매칭하기 전에 backtracking이 일어나는 경우, 이 backtracking이 일어났던 vertex가 해당 query에서 매칭의 성공 여부를 결정하는 핵심 vertex일 가능성이 존재한다. 따라서 각 vertex마다 backtracking이 일어난 횟수를 기록하여 많이 일어난 vertex일수록 더 우선순위가 높아지도록 하였다.

**Perform backtracking**

* 위의 matching order에 따라 현재 매칭할 vertex u를 선택한다.
* u의 extendable candidate set에 있는 각 v에 대해 매칭을 진행한다.
* 해당 v와 연관된 extendable candidate set size를 update한다. (이 과정에서 empty extendable candidate set이 있는지 확인한다)
* recursive하게 매칭을 진행한다.
* 더 이상 vertex가 선택되지 않는 경우 결과를 출력한다.

**Run**

Environment: Ubuntu (WSL2)

Compile and Execute: (same as reference)

mkdir build

cd build

cmake ..

make

./main/program <data graph file> <query graph file> <candidate set file>